ARCHITECTING HIGH-PERFORMANCE, EFFICIENT, AND SCALABLE HETEROGENEOUS MEMORY SYSTEMS WITH 3D-DRAM

A Dissertation
Presented to
The Academic Faculty

By

Chiachen Chou

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2017

ARCHITECTING HIGH-PERFORMANCE, EFFICIENT, AND SCALABLE HETEROGENEOUS MEMORY SYSTEMS WITH 3D-DRAM

Approved by:

Dr. Moinuddin Qureshi
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Sudhakar Yalamanchili
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Hyesoon Kim
College of Computing
Georgia Institute of Technology

Dr. Aamer Jaleel
NVIDIA Research
NVIDIA

Dr. Tushar Krishna
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: March 16, 2017

Dedicated to my family

ACKNOWLEDGEMENTS

I cannot begin to express my deepest gratitude to my advisor, Dr. Moinuddin Qureshi, for his guidance and advice throughout my graduate school career. As his words of wisdom often provide me invaluable insights, his lines of thoughts often inspire me to improve in finding new ideas, investigating novel solutions, and communicating the ideas effectively. Also, I am deeply indebted to Dr. Aamer Jaleel, who is equally important to me in my PhD student life. Besides a handful of publications that we collaborate on and a couple of opportunities that I work with him as an intern, I look up to him because of his always-positive mentality against adversity and his caring attitude for people. His advice, which helps me break through the ups and downs, will always be sought after during my career.

I would like to express my appreciation to my committee members, Dr. Sudhakar Yalamanchili, Dr. Hyesoon Kim, and Dr. Tushar Krishna, not only for their feedback and advice that improve the quality of my thesis but also for the knowledge and the passion that I learn from them. Dr. Yalamanchili taught a very insightful network-on-chip class that I learned the importance of clear communication and the influence of a well-organized lecture. Dr. Kim taught the advanced computer architecture class that I learned many of the fundamental computer architecture concepts such as branch prediction and parallel architectures. I am also fortunate to have the wonderful chance of being an colleague with Dr. Krishna when I was interning at Intel Massachusetts. His enthusiasm and attention to details often inspire me to keep improving forward.

I must also thank my labmates, Prashant Nair, Jian Huang, Vinson Young, Swamit Tannu, Dae Hyun Kim and Gururaj Saileshwar, for being my partners and supporters in this journey. Our time spent together to discuss research ideas or to stay up late for conference submissions is non-volatile pieces of memory to me. I enjoy the stories of life that my colleagues share with me and also the bond with them as a big family. I am also thankful to have other fellow PhD students at Georgia Tech that have provided their help and support.

My heartfelt thanks is for Nicholas Tzou, who helped me settle down at my earliest life in Atlanta by letting me crash on his couch. I also want to thank Ted Chang, whom I know since my college and share joy and tear with. Besides, I also want to thank Ching-Kai Liang, also a PhD student working in computer architecture, for his opinion and the time we spent together.

In my PhD career, I also had a few opportunities to work as an intern in industry-leading companies. I very much appreciate Dr. Kermin Fleming and Dr. Joel Emer for their mentorship. They had looked after me when I did my first internship at Intel. I also thank Dr. Evgeny Bolotin and Dr. Alex Ramirez for their guidance and supervising effort during my second internship at NVIDIA. I learned a great lesson from them. I also had a great pleasure of working with Rajat Agarwal and Dr. Suresh Chittor when I interned at Intel Hillsboro. Their sensitivity and decision-making skills as computer architects are beyond exemplary.

The completion of my PhD would not have been possible without my parents' unconditional support. My parents, Cheng-Lin Chou and Fang-Ling Huang, raise me to stay hungry to learn, prepare me with the skills for challenges, and motivate me to pursue the knowledge. My brother, Wei-Chen Chou, is my role model, who teaches me with enormous patience and shapes me to who I am; he provides the warmest support so that I am able to achieve my goals. Without their love, I could not have the accomplishment. Finally, my highest appreciation belongs to my wife, Yungchi Fan, a lovely and considerate lady that I respect highly. Your companion always makes the hard time easier and the happy time merrier. Throughout the years, it is you, who strengthens and inspirits me with your brightness and kindness. Because of you, my growth is beyond words, and I share the achievement with you. I love you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Advancements in packaging technology enable high-bandwidth 3D-DRAM that mitigates the memory bandwidth wall. With the integration of 3D-DRAM and high-capacity memory, heterogeneous memory systems are able to satisfy the high memory bandwidth demand of processors. However, traditional management techniques developed for on-chip caches are neither suitable nor efficient for heterogeneous memory systems with high-bandwidth 3D-DRAM. This dissertation investigates the problems of managing heterogeneous memory systems and proposes simple architectural solutions that improve the performance of such systems.

The management of caches requires several operations to maintain the functionality of the cache; such operations in DRAM caches consume bandwidth that could have been used by critical requests. This dissertation proposes three techniques: bandwidth-aware bypass, DRAM cache presence, and neighboring tag cache. All three techniques combined reduce the bandwidth bloat of secondary operations by 32%, reducing the execution time by 10%.

The DRAM-cache hit rate for PCM-based heterogeneous memory systems is critical to mitigating long PCM access latency. Although set associativity can improve the hit rate, it can degrade performance by as much as 20% because of the latency overhead. The dissertation proposes a morphable set-associative cache with two parts: (1) an infrastructure that enables a low-cost transition between two degrees of set associativity and (2) a mechanism that dynamically chooses the highest-performance set associativity. The proposed techniques limit performance degradation to less than 1.5%.

Conventional techniques manage data for heterogeneous memory systems such that 3D-DRAM services all the requests. The dissertation shows that such an approach underutilizes bandwidth provided by commodity DRAM and that performance can improve by maximizing aggregate system bandwidth. It proposes a simple mechanism that explicitly controls data movement between the two DRAM modules, distributes memory accesses

proportionally to the respective bandwidth, and improves performance by 11% in the cache mode and 10% in the flat mode.

This dissertation addresses the dilemma: We must select either two-level memory or the DRAM cache. While the two-level memory has 3D-DRAM capacity, the DRAM cache has software transparency and fine-grained data transfer. To address this dilemma, this dissertation proposes a cache-like memory organization with all benefits. The proposed technique uses line swapping for locality and has a line location table and a location predictor that eliminate the overhead associated with locating lines. Overall, the proposed technique improves performance by an average of 78%, very close to an idealized system with 82% improvement.

This dissertation also shows that in multi-socket systems, coherent DRAM caches can outperform the current design, the memory-side cache; however, enabling coherent DRAM caches encounters two key challenges: a large coherence directory and slow request-for data operations. The dissertation proposes a DRAM-cache coherence buffer that addresses the large coherence directory problem and a sharing-aware bypass mechanism that reduces the latency overhead of request-for-data operations. The proposed techniques incur negligible overhead and reduce execution time by 25% on average, which is within 5% of the performance of an idealized coherent DRAM caches.

CHAPTER 1
INTRODUCTION

Over past several decades, the speed of processor has improved significantly while the speed of memory systems has not continued at the same pace [1]. This performance disparity between processors and memory systems deteriorates as the compute paradigm shifts towards enormous parallelism by exploiting instruction-level parallelism, vectorization, and multi-threaded and multi-programmed applications. In the parallel compute paradigm, workloads concurrently access a large amount of data, so system performance depends on the throughput of memory systems. The throughput is determined by memory bandwidth, which becomes the critical performance bottleneck. Such a bottleneck is also referred to as the *memory bandwidth wall* [2, 3]. Therefore, for a memory system to be high performance, it must provide high bandwidth.

1.1 The Problem: Managing Heterogeneous Memory Systems with 3D-DRAM

The primary building block of memory systems today is dynamic random access memory (DRAM) [4, 5], which is used in the form of dual in-line memory module (DIMM). The bandwidth of DIMM-based DRAM is limited by the pin count, so it is not able to satisfy the growing demand of memory bandwidth. To overcome such a constraint, memory manufacturers have enhanced the DRAM technology by die-stacking multiple DRAM modules in a chip and packaging the 3D-stacked DRAM chips with processors. Examples of such emerging technology, referred to as *3D-DRAM*, include Hybrid Memory Cube (HMC) [6, 7, 8], High Bandwidth Memory (HBM) [9], and Wide I/O (WIO) [10]. Recent announcements of products with 3D-DRAM show that 3D-DRAM provides 4-8X bandwidth as DIMM-based DRAM; however, its capacity is in the range of only a few gigabytes, smaller than a few tens of gigabytes provided by commodity DIMM-based DRAM.

As 3D-DRAM cannot fully replace the commodity DRAM in a cost-effective manner, future memory systems are likely to consist of high-bandwidth 3D-DRAM and high-capacity DIMM-based DRAM. Such a memory system, referred to as a *heterogeneous memory system*, is the focus of this dissertation.

The key question for computer architects is how to architect a heterogeneous memory system with high-bandwidth gigabyte-capacity 3D-DRAM. Although prior studies have attempted to address the challenge of managing the system, they are based on conventional techniques developed for on-chip static random access memory (SRAM) caches. As high-bandwidth 3D-DRAM exhibits characteristics that differ from those of the small SRAM caches, prior architectures are neither suitable nor effective for heterogeneous memory systems; they fail to maximize the benefits provided by 3D-DRAM and thus deliver sub-optimal performance. Such a performance gap necessitates architectural solutions tailored for heterogeneous memory systems with high-bandwidth 3D-DRAM. This dissertation investigates the deficiencies of conventional techniques and proposes simple and effective architectural innovations that improve performance. The dissertation develops the techniques at various levels of the memory system: managing only 3D-DRAM, coordinating 3D-DRAM and DIMM-based DRAM, and scaling the system to multiple nodes. The following sections discuss the three problems.

1.1.1 Architecting the 3D-DRAM as a Hardware-Managed DRAM Cache

One way to use the 3D-DRAM is to architect it as a hardware-managed cache, referred to as a *DRAM cache*, which is an intermediate level between on-chip caches and memory. To accommodate the large size of tag storage, current designs place tags in 3D-DRAM. A DRAM cache uses the available 3D-DRAM bandwidth not only for critical data on cache hits but also for other secondary operations such as cache miss detection, fill on cache miss, and writebacks from the on-chip cache. Ideally, only critical cache hits should consume the 3D-DRAM bandwidth, but secondary operations in the current DRAM cache

design consume 2.8X as high bandwidth as their critical counterpart. To address the problem, the dissertation proposes bandwidth-efficient architecture (BEAR) for DRAM caches that reduces the bandwidth consumption of secondary operations. BEAR integrates three components, each targeting one source of the following operations: miss detection, miss fill, and writeback detection.

The dissertation also investigates the problem of DRAM caches in a heterogeneous memory system with *non-volatile memory*, such as phase-change memory (PCM). PCM has 4-8X as much capacity as but has longer latency than the commodity DRAM; therefore, DRAM cache misses suffer from long PCM access latency. As the previously proposed designs use a direct-mapped cache organization, one way to mitigate such a latency overhead is to improve the DRAM cache hit rate by employing set associativity. For example, a two-way cache in which a memory request accesses both ways and consumes higher bandwidth. The dissertation shows that the two-way cache always increases cache hit latency because of extra bandwidth consumption but not always improves the cache hit rate. The dissertation proposes morphable set-associative DRAM cache (MOSAIC) that uses set associativity only when it is useful and that dynamically chooses the best-performance set associativity with low-cost transition overhead.

1.1.2 Coordinating 3D-DRAM and DIMM-Based DRAM

The second part of the dissertation addresses the challenge of coordinating both 3D-DRAM and DIMM-based DRAM. The first problem is utilizing the aggregate system bandwidth; conventional designs tend to maximize the number of memory accesses serviced by 3D-DRAM. When the commodity DRAM bandwidth is a significant fraction of the overall system bandwidth, prior techniques inefficiently utilize the total system bandwidth and thus yield sub-optimal performance. In such situations, performance can be improved if memory accesses are distributed proportionally to the respective bandwidth of each memory. The dissertation proposes bandwidth-aware tiered-memory management (BATMAN),

a runtime mechanism that achieves the desired access distribution without requiring significant hardware or software support.

Besides the resource management of bandwidth, the utilization of capacity is also a challenge. In a DRAM cache, the capacity of 3D-DRAM is not visible to operating systems (OS) while two-level memory that uses 3D-DRAM as part of memory space increases the effective memory capacity but requires OS support to migrate data at a page granularity. To benefit from both designs, the dissertation proposes a hardware-based cache-like memory organization (CAMEO) that not only makes 3D-RAM visible as part of the memory address space but also exploits data locality on a fine-grained basis. CAMEO retains recently accessed data in 3D-DRAM and swaps out the victim line to the commodity DRAM. The physical location of a cache line changes at run-time, so CAMEO uses a low-overhead line location table and an accurate line location predictor to avoid the serialization overhead of table lookups and memory accesses.

1.1.3 Scaling to Multi-socket Systems

The third part of the thesis investigates the scalability problem of heterogeneous memory systems with multiple DRAM caches in multi-socket systems. The current design, the memory-side cache (MSC), restricts the DRAM cache to keeping only local data and relying on small on-die caches for remote data. The implicitly coherent MSC obviates the need for coherence support, but it suffers from long inter-node latency overhead on every on-die cache miss that accesses the remote data. A desirable alternative is to allow the DRAM cache to retain both the local and the remote data. However, as data blocks can be located in multiple DRAM caches, this design, referred to as *coherent DRAM cache (CDC)*, requires coherence support for correctness. Enabling the giga-scale CDC encounters two key challenges: First, the coherence directory can be as large as a few tens of MBs. Second, cache misses that access the read-write shared data cause longer delays because they must access the DRAM cache. To address both problems, this dissertation proposes DRAM

cache for multi-node systems (CANDY), a low-cost and scalable solution that consists of two techniques: a two-level structure that mitigates the overhead of latency and storage of the coherence directory; and a sharing-aware bypass mechanism that forces read-write shared data to be stored only in on-chip caches.

1.2 Thesis Statement

Conventional techniques developed for on-chip caches are ineffective for managing high-bandwidth 3D-DRAM; simple architectural innovation that addresses the challenge of hardware management, resource utilization, and scalability can improve the performance of heterogeneous memory systems.

1.3 Contributions

This dissertation makes the following contributions.

- **Hardware Management:** The dissertation shows that the reduction in DRAM cache bandwidth consumption significantly improves cache performance and proposes a bandwidth-efficient architecture for DRAM caches. Also, the dissertation shows that with PCM-based memory, increasing the set associativity of the DRAM cache degrades performance if the hit rate improvement is small and proposes a morphable set-associative DRAM cache that dynamically uses the desired set associativity.

- **Resource Utilization:** The dissertation presents a simple and effective bandwidth management technique that achieves access distribution proportional to the respective DRAM bandwidth with negligible overhead. The dissertation also shows neither two-level memory nor a DRAM cache provides optimal performance for a large set of workloads. To address the problem, the dissertation proposes a hardware-based mechanism referred to as cache-like memory organization, which not only increases the effective memory capacity but also exploits data locality on a fine-grained basis.

- **Scalability:** The dissertation analyzes two DRAM-cache architectures (memory-side caches and coherent DRAM caches) for multi-socket systems and observes that coherent DRAM caches outperform memory-side caches. To enable giga-scale coherent DRAM caches, the dissertation proposes a low-cost architecture that addresses the challenges of coherent DRAM caches: a large coherent directory and slow cache misses that access read-write shared data.

1.4 Thesis Organization

The rest of the dissertation is organized as follows. Chapter 2 provides background on prior work and related studies. Chapters 3 and 4 present the hardware management of DRAM caches for bandwidth efficiency and dynamic adaptability, respectively. While Chapter 5 addresses the challenge of bandwidth management for heterogeneous memory systems, Chapter 6 addresses the challenge of capacity management. Chapter 7 shows a scalable architecture for multi-socket systems and Chapter 8 summarizes the dissertation and discusses future work.

CHAPTER 2

RELATED WORK

As the need to address the memory wall becomes imminent, a huge body of studies in the computer architecture community aims to mitigate the gap between processors and memory. This chapter, presenting prior studies related to heterogeneous memory systems, is organized as follows: (1) 3D-DRAM; (2) two-level memory in prior work; (3) DRAM caches; (4) coherence protocol and improvement; and (5) other techniques that improve memory systems.

2.1 3D-DRAM

An emerging memory technology that provides higher memory bandwidth is 3D-DRAM, which stacks multiple layers of DRAM modules in a single package. 3D-DRAM uses through-silicon vias (TSVs) [11, 12], which connect two chips vertically using a copper metal to overcome the limits of conventional interconnects. This technology breakthrough offers many benefits, including lower active and dynamic power and higher I/O speed. Also, technology advancement in chip packaging enables the placement of 3D-DRAM that is close to processors and increases the number of pins that connect 3D-DRAM and processors. Therefore, DRAM chip stacking and packaging allow 3D-DRAM to provide high memory bandwidth. While current Joint Electron Device Engineering Council (JEDEC) specifies the high bandwidth memory (HBM) standard [9], memory vendors also provide other types of 3D-DRAM, which includes Hybrid Memory Cube (HMC) [6, 7, 8] and wide I/O (WIO) [10]. Examples of products with 3D-DRAM, which is packaged and integrated with processors, are AMD Zen [13], Intel Xeon Phi [14], and NVIDIA Pascal [15].

Although 3D-DRAM provides high memory bandwidth, the technology is still maturing; its capacity is smaller than that of commodity DIMM-based DRAM. F igure 2.1 shows

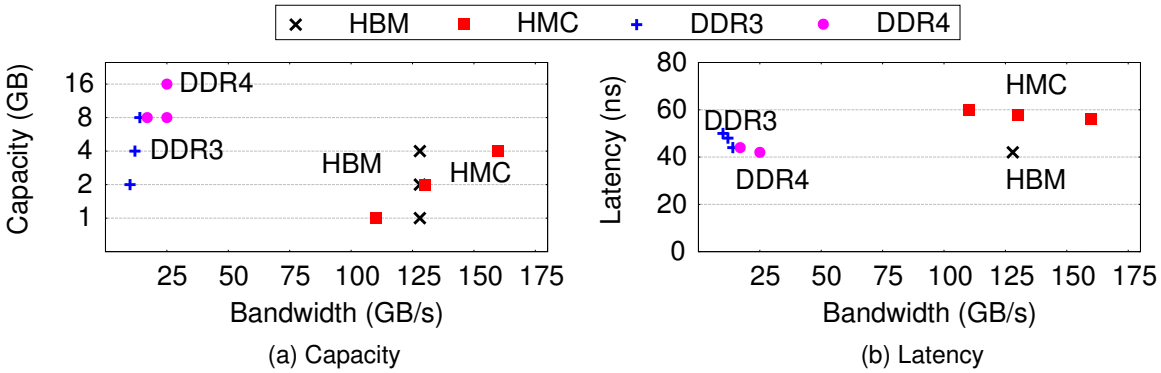

Figure 2.1: Comparison of Capacity, Latency and Bandwidth among DDR3, DDR4, High Bandwidth Memory (HBM), and Hybird Memory Cube (HMC).

a comparison of different DRAM technologies. Figure 2.1(a) shows the capacity as a function of provided memory bandwidth. Although 3D-DRAM provides two to eight times as much bandwidth as conventional DRAM, such as DDR3 and DDR4 [4, 5], the capacity of 3D-DRAM is still in the range of a few gigabytes while the memory capacity of DDR DRAM is as large as a few tens of gigabytes. As the memory capacity is a key factor of the memory system, 3D-DRAM cannot fully replace DDR DRAM in a cost-effective manner. In addition, because 3D-DRAM and commodity DDR DRAM use the same DRAM cell technology, which determines the access latency of DRAM, the access latency of both 3D-DRAM and DDR DRAM is similar, shown in Figure 2.1(b). Therefore, 3D-DRAM does not have a latency advantage over DDR DRAM.

2.2 Two-Level Memory

As 3D-DRAM has a relatively small capacity, it is likely to co-exist with DIMM-based DDR DRAM, so the two DRAM technologies form a heterogeneous memory system. For this system, with two memory components–high bandwidth 3D-DRAM and high-capacity DDR DRAM–computer architects use the 3D-DRAM in two ways: as a part of the OS-visible memory or as a hardware-managed DRAM cache, shown in Figure 2.2. This section first introduces prior work on using 3D-DRAM as part of the OS-visible memory, and the next section discusses studies that use 3D-DRAM as a hardware-managed cache. When

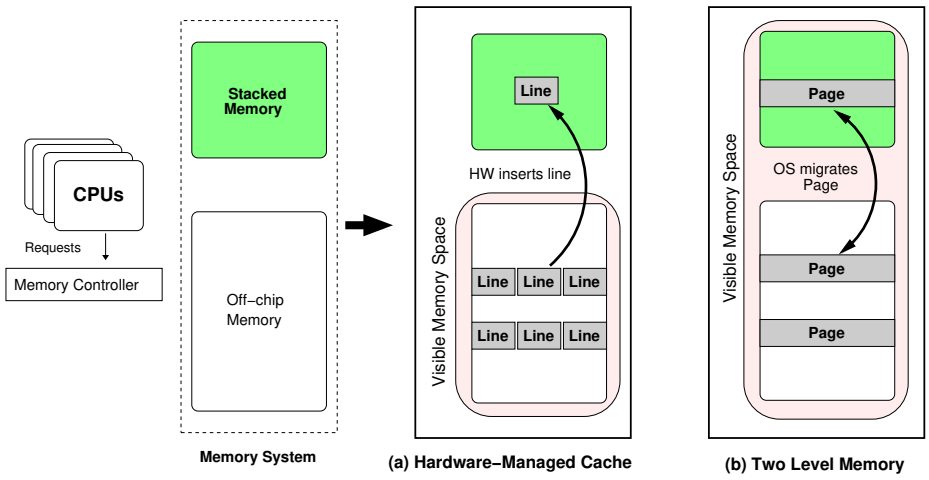

Figure 2.2: Architecture for using 3D-DRAM in memory systems: (a) Hardware-managed cache at line granularity (b) OS-managed two-level memory at page granularity (Note: Figure not to scale).

3D-DRAM is used as part of the memory space, the total memory capacity of the system is the sum of the capacity of 3D-DRAM and commodity DRAM. The system is referred to as two-level memory, which has two memory technologies of distinct characteristics.

When heterogeneous memory systems with 3D-DRAM are used as two-level memory, pages that reside in 3D-DRAM are serviced with high bandwidth while pages in commodity DRAM are serviced with low bandwidth. The data placement between two memory components determines the performance of the system; a high-performing system should place frequently accessed data in the 3D-DRAM. Prior work has proposed either a hardware-software coordination approach or software-only solutions. Sim et al.[16] use hardware counters to identify hot data segments (i.e., frequently accessed data segments) in the memory system and moves these hot segments to the 3D-DRAM. Meswani et al. [17] also use the OS to identify hot memory pages in the system and move t hem to the 3D-DRAM. Although both designs focus on how to identify hot data in the memory in a cost-effective manner, the granularity of data transfer in both proposals (1KB and 4KB) is larger than the typical size of the data transfer of memory requests, which is 64B.

Also, prior studies that investigate a two-level memory system with distinct memory characteristics (e.g., memory latency). originate from mutli-socket systems, which consist

of many computing nodes. Each node has computation capability and also memory capacity, and all nodes are connected through a long-latency interconnect. Processor on any node can access the memory on its home node, or on any other nodes, but memory requests that go to other nodes incur higher latency. Therefore, the latency of a memory request depends on which node the memory request goes; such non-uniformity in memory latency suggests the name "non-uniform memory access". For this kind of system, the objective is the management of data placement and migration to avoid latency overhead of remote data access. While Dahlgren and Torrellas propose cache-only memory architectures (COMA) [18], Falsafi and Wood propose reactive NUMA that unifies COMA and NUMA [19] For other type of two-level memory, Machanick proposes RAMPage, which uses SRAM storage on chip as part of memory space to mitigate the speed difference in CPU and DRAM [20].

2.3 DRAM Caches

3D-DRAM can be architected not only as part of the OS-visible memory space but also as a hardware-managed cache, an intermediate level between on-chip SRAM caches and the main memory. The dissertation refer to the hardware-managed cache as *DRAM cache*, which does not require software modification for data movement. The appealing advantage has attracted many research proposals in the last few years in the computer architecture community and also the adoption of the DRAM cache in commercial products [14]. Recent studies have addressed three key challenges of the architecture of the DRAM cache: (1) tag storage, (2) the block size, and (3) set associativity. This section describes recent DRAM cache designs that have appear at top conferences and compares their strengths and weaknesses. In addition to the academic DRAM cache proposals, this section also describes a recently disclosed DRAM cache, designed by the industry vendor. Table 2.1 shows the comparison of different DRAM cache designs. The table compares each design on the tag storage, SRAM overhead, cache block size, and the set associativity of the cache. (The SRAM storage overhead is with respect to a 1GB DRAM cache.)

Table 2.1: Comparison of DRAM cache designs. The SRAM storage overhead is calculated based on 1GB DRAM cache.

Design	Tag Stored	SRAM Overhead	Cache Block Size	# of Ways
Loh-Hill	DRAM	6MB	64B	29-way
Mostly-clean	DRAM	7KB	64B	29-way
Alloy	DRAM	2KB	64B	Direct-mapped
Footprint	SRAM	6MB	1KB Sector, 64B Line	8-way
Unison	DRAM	200KB	1KB Sector, 64B Line	4-way
Bimodal	DRAM	400KB	64B and 512B	8-way
Tag table	SRAM	4MB	64B	64-way

Loh and Hill [21] propose the earliest DRAM cache design, which uses 3D-DRAM technology, referred to as Loh-Hill cache. To exploit the row-buffer hit in DRAM, the Loh-Hill cache stores tags along with data in the same row buffer of 3D-DRAM. Each row buffer stores 32 64B lines, 29 of which are data lines, forming a 29-way set, and three of which are tag storage lines for the 29 data lines. To read data in the DRAM cache, a request first reads the tags (three lines) to determine the location information of the requested line and accesses the data line (one line). Thus, a serialization between the tags and the data incurs high latency for each access to the DRAM cache. For data that are not in the DRAM cache, the Loh-Hill cache uses a tracking structure, referred to as MissMap, that avoids the tag look-up process. The tracking structure requires a 6MB SRAM storage for a 1GB DRAM cache, which is typically over the allowance in current processor chips. Therefore, the Loh-Hill cache is neither scalable nor practical.

To avoid the SRAM storage overhead, the computer architecture community have dedicated an enormous amount of effort such that many ideas appear in top conferences. Based on the Loh-Hill cache, Sim et al. [22] propose a mostly-clean cache to address the SRAM storage overhead of the Loh-Hil Cache. Instead of the MissMap structure, which keeps track of cache lines in the DRAM cache, the mostly-clean cache predicts if the cache line is present in the DRAM cache. If a line is a predicted miss, a memory request that goes to off-chip DRAM is issued in parallel. However, the authors did not address the serialization problem of a hit. Also, others have applied developed ideas for SRAM caches to designing

DRAM caches. For example, use the sector cache to reduce the tag size [23]. Jevdjic et al. [24] propose *footprint cache*, which uses a 1KB sector to minimize the tag storage of SRAM. They also equip the DRAM cache cache with a memory footprint prefetcher [25]. Although the use of the sector cache reduces the tag storage, it is not scalable. To address the scalability problem, the authors extend their work and propose the *unison cache* [26], a four-way set-associative sector cache that stores tags in 3D-DRAM. To address the serialization issue of tag look-ups, the unison cache uses a way predictor [27], which speculates the location of the requested cache line.

Franey and Lipasti [28] propose a tag storage mechanism, referred to as *tag table*, which is inspired by page table in the OS memory management. The tag table compress tags to minimize the size of tags, and use part of the on-chip last-level cache to cache the compressed tag entries. However, for a 1GB DRAM cache, the tag table still requires up to 4MB for the tag arrays. Instead of use 64B cache line, other DRAM cache design uses large cache line such as 512B or 1KB, which reduces the overhead of tag storage. Gulur et al. [29] propose the *bimodal cache*, which uses two cache-line sizes in a DRAM cache to exploit spatial locality. The two cache-line sizes are 64B and 512B; the bimodal cache dynamically selects the granularity of the block size for an individual memory address. In addition, the authors also use a block-size predictor and a small SRAM structure to cache tags in the SRAM to speed up the tag look-up process.

To optimize the DRAM cache hit latency, Qureshi and Loh [30] propose the *alloy cache* that organizes the DRAM cache as a direct-mapped cache and places tags and data together. In a direct-mapped cache, every data line can be in only one location, which precludes the need to resolve the location before accessing the data. The tag and the data are bundled such that one 3D-DRAM access bursts out one unit of the tag and the data. Therefore, data that are in the DRAM cache (i.e., cache hits) require only one 3D-DRAM access, which significantly lowers the hit latency (no serialization, no extra accesses). For the miss latency, the alloy cache uses a miss predictor to predict if a cache line is likely to miss in

the DRAM cache. If so, a request is issued to the off-chip memory in parallel to save the serialized look-up time. However, the direct-mapped property degrades the DRAM cache hit rate. Based on the recently disclosed document [14], the industry vendor employs a DRAM cache design that is similar to the alloy cache. The DRAM cache is "a direct-mapped cache with 64-byte cache lines." Also, the commercial product places the tags "in the error-correcting code bits corresponding to the cache lines." [31]

2.4 Multi-socket System: Coherence Protocol and Shared Memory Systems

2.4.1 Coherence Directory and Protocols

To address the scalability issue of memory systems, coherence protocol is a popular research area in the past few decades [32]. For scalability, multiprocessor and multi-node systems typically adopt directory-based protocols [33, 34, 35, 36]. For directory-based protocols, the key challenge is to design a low-cost coherence directory [37, 38, 39]. One approach is duplicate-tag directory, proposed by Barroso et al. [40]. Although duplicate tag directory incurs low area cost, but its high associativity makes it energy inefficient and hard to scale. Also, a similar approach is adopted in commercial products [41].

Sparse Directory is one of the appealing directory-based protocol, because of its scalability and energy-efficiency. Prior studies focus on reducing the directory storage overhead by reducing the sharer vector length or by reducing the conflict misses. Ferdman et al. [42] propose *cuckoo directory*, which uses cuckoo hash table to reduce conflict misses of sparse directory, thus reducing the storage overhead. Cuesta et al. [43] propose mechanism that deactivates coherence for private memory blocks to increase the effectiveness of sparse directory. Sanchez et al. [44] propose a sharer set entry for the scalability of sparse directory. Demetriades and Cho [45] propose *stash*, allowing private blocks to be valid in the cache without corresponding directory entries. Meanwhile, Acacio et al. [46] use compression technique to place recently accessed directory entries in a fast cache to mitigate the latency. Valls et al. [47] propose separate directories for shared and private data.

2.4.2 Shared Memory Systems

One of the problem in a shared-memory computer system that has non-uniform memory access latency is the long-latency and low-bandwidth inter-socket network access. As the network is a scarce resource, many prior works, such as remote memory operations, focus on how to reduce the network traffic by either hardware-based or software-based and by either delegation or privatization. Zhang et al. [48] propose a mechanism that executes atomic synchronization operations at the memory controller of the home node, to which the synchronization variable belongs. Ann et al. [49] propose *scatter-add*, which uses vector scatter for fetch-and-add operations Hoffmann et al. [50] propose a low-overhead technique that implements remote store operations. Zhang et al. [51] exploit the communtativity to reduce the update overhead for shared data in the system. Other work related to the shared memory system is shown in Section 2.2. Specifically, as opposed to line-granularity coherent caches in a NUMA system (Cache-Coherent Non-Uniform Memory Architecture, or ccNUMA), COMA operates at a page granularity, and relies on the operating system to maintain coherence.

2.5 Other Techniques to Improve Memory Systems

In the past decades, memory systems have gained a tremendous amount of attention from the computer architecture community. While some focus on cache replacement policy, cache partition, and data prefetch, others focus on DRAM scheduling and memory resource management. This section introduces prior work on each related topic.

2.5.1 Cache Replacement and Bypass Policy

One approach that improves the cache hit rate and reduces cache misses is cache replacement and bypass policies, which reduce the main memory bandwidth consumption and improve the bandwidth efficiency. Independently, both Qureshi et al. [52] and Jimenez [53]

find that the least recently used replacement policy does not account for the memory access pattern of workloads; they address the problem by an adaptive insertion or promotion policy based on dynamic adjustment or static profiling. Another cache replacement policy, proposed by Jaleel et al. [54] use a re-reference interval prediction (RRIP) that tracks the reuse characteristics of cache lines and evicts cache lines that are least likely to be reused. Wu et al. [55] extend the RRIP work and uses a signature of each cache line to predict the cache-line reuse behavior. Equally important studies that improve cache performance are cache bypass policies, which identify cache blocks that are never reused after being installed in the cache. Lai et al. [56] use the dead block information to prefetch data from main memory. Khan et al. [57] propose a technique that predicts whether a cache block is dead and chooses to evict such blocks early.

2.5.2 Resouce Management

One approach that maximizes the performance of the system and also ensures the fairness among programs is the management of memory system resources, including cache partition, memory scheduling, and memory partition. Cache partition, proposed by Qureshi and Patt [58], divides the available cache capacity based on the behavior of each program and satisfies the demand of each program. To ensure the fairness of the workloads and avoid starvation in cache partitioning, Zahedi et al. [59] use the concept of the market equilibrium to optimize other metrics such as fairness and sharing incentive. Besides the resource management of cache partition, a large body of prior work focuses on memory scheduling, which manages the resource in the main memory level. One example of memory resource management, proposed by Rixner et al. [60], is memory scheduling, which exploits row buffer hits in DRAM and reorders the memory requests to serve them faster. Furthermore, as many programs running on the same chip are sharing the memory resources, the memory controller manages the scheduling policy to maximize memory throughput and to prevent starvation in the system [61, 62].

CHAPTER 3

BANDWIDTH-EFFICIENT ARCHITECTURE FOR DRAM CACHES

This chapter investigates the problem of bandwidth use in DRAM cache. Unlike SRAM caches that has a large set of wires for access bandwidth, the DRAM cache uses the DRAM memory controller with a narrow bus to access tags and data and thus has limited bandwidth. Two recent designs for DRAM cache are shown in Figure 3.1. One key attribute of these two designs is the tag placement. As giga-scale DRAM caches hold millions of cache lines, the tag storage can be as large as few tens of MB. These two design place the tags in the 3D-DRAM to avoid incurring a SRAM tag storage overhead. Figure 3.1(a) shows a design, proposed by Loh and Hill [21], which exploits the DRAM row buffer hit by placing tags and data in a single DRAM row buffer. This design, referred to as *Loh-Hill cache*, organizes the cache as a 29-way set-associative cache and stores the tags for the 29 ways in the first three cache lines of the 2KB row buffer. Servicing a cache hit requires first reading and checking the tags from the DRAM row buffer and then reading the data from the DRAM row buffer (on a tag match). Therefore, the Loh-Hill cache incurs not only serialization overhead between tag look-up and data access but also four times bandwidth overhead for every cache request.

To address the serialization overhead, Qureshi and Loh [30] propose *Alloy cache*, shown in Figure 3.1(b). Alloy cache places the tags and data together to form a *tag and data (TAD* unit and organizes the cache as a direct-mapped cache. Every cache request reads one TAD, which is 72 bytes. In a direct-mapped cache, data can be in one location; thus, alloy cache avoids the serialization overhead and also the additional bandwidth consumption for three tag lines. Compared to the Loh-Hill cache, which consumes 256B for a cache request, alloy cache reads only 72B for each request, significantly reducing the bandwidth consumption. As the alloy cache is more bandwidth efficient, this dissertation

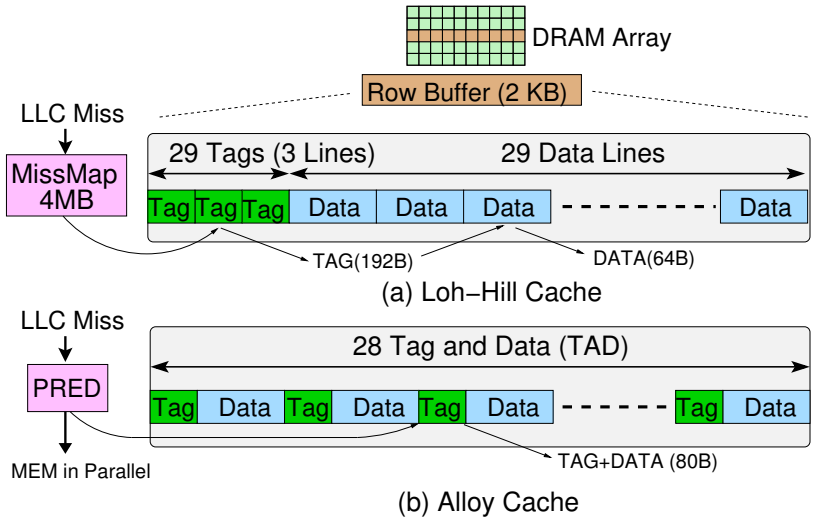

Figure 3.1: Recent Designs for DRAM Cache: (a) Loh-Hill Cache transfers 3 lines for tag and 1 line for data on each hit (256 bytes), and (b) Alloy Cache transfers 1.25 cache line for each hit (80 bytes).

assumes the alloy cache as the baseline DRAM cache model. However, the state-of-the-art alloy cache design still suffers from bandwidth bloat problem.

3.1 Motivation: Bandwidth Bloat in DRAM Caches

The conventional approach of using DRAM to architect main memory follows a simple request response protocol. When an address misses in the on-chip last-level cache (LLC), the memory controller fetches the data from the DRAM devices. With DRAM configured as main memory, the effective raw DRAM bandwidth is the total number of bytes transferred on the data bus (i.e., #LLC_misses * LLC_Line_Size). Architecting DRAM as a cache, in contrast, requires additional bandwidth to support cache functionality (e.g., cache fills, cache probes). This dissertation proposes a metric termed *bloat factor*, which is defined as the total bytes transferred on the 3D-DRAM data bus divided by the total bytes required to satisfy all LLC misses, as shown in Equation 3.1.

$$BloatFactor = \frac{\sum TotalBytesTransferred}{\sum UsefulBytesTransferred} \tag{3.1}$$

Ideally, the bloat factor value should be 1, meaning that the entire DRAM cache bandwidth contributes to servicing LLC misses. However, as Figure 3.2(a) illustrates, bloat factors are 7.3X and 3.8X for Loh-Hill and alloy caches, respectively. As DRAM cache hit latency comprises two parts: DRAM array access latency and queuing delay, the bandwidth bloat increases DRAM service time by increasing the queuing delay. Shown in Figure 3.2(b), the DRAM cache hit latency is 409 cycles and 239 cycles with respect to Loh-Hill and alloy cache, while an ideal case (termed Bandwidth-Optimized cache (BW-Opt)) that all secondary operations are free has DRAM cache hit latency of only 97 cycles. Therefore, BW-Opt reduces the L4 hit latency significantly, and thus outperforms both Loh-Hill and alloy cache, as shown in Figure 3.2(c).

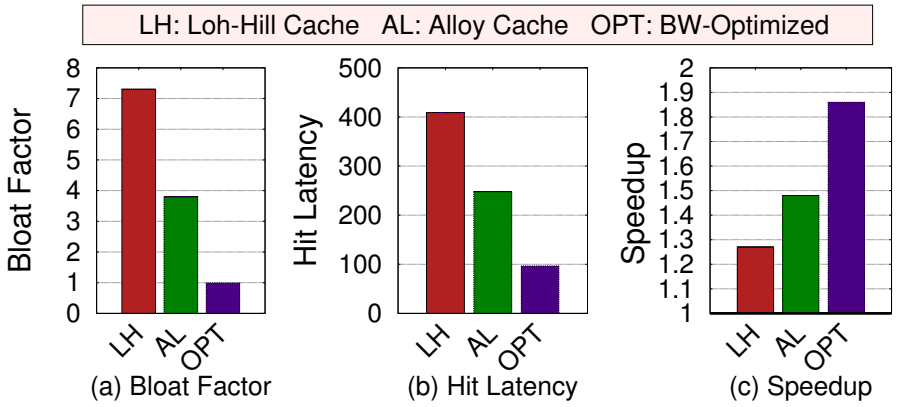

Figure 3.2: Comparison of Loh-Hill (LH), Alloy (AL), and Bandwidth- Optimized (OPT) cache: (a) Bloat Factor, (b) Hit Latency, and (c) Speedup with respect to no DRAM cache.

3.1.1 Breakdown: Where Does the Bandwidth Go?

Bandwidth bloat in DRAM caches corresponds to the steps in implementing cache functionality. Unlike memory which only holds data, DRAM caches hold both tag and data. Typically, on read requests, a tag is used to determine if an address exists in the cache. Thus, every cache lookup requires both tag and data to be fetched from the DRAM cache. If the cache lookup results in a hit, the first source of bandwidth bloat (referred to as *Hit Probe*) can be attributed to tag fetch (the data is critical and hence not a bandwidth bloat).

If the cache lookup results in a miss, the second source of bandwidth bloat (referred to as *Miss Probe*) can be attributed to the fetching of both tag and data.Typically, a cache miss requires inserting a line into the cache. Thus, the third source of bandwidth bloat (referred to as *Miss Fill*) can be attributed to filling the new tag and data into the DRAM cache.

In addition to read requests, the processor can return dirty data from the on-chip LLC by issuing writeback requests. On a writeback request, the DRAM cache must be consulted to determine whether the corresponding line already exists in the DRAM cache. Should the line exist in the DRAM cache, the DRAM cache contents must be updated for correctness. Thus, the fourth source of bandwidth bloat (referred to as *Writeback Probe*) can be attributed to fetching the tag to detect whether or not to update the DRAM cache contents. If the Writeback Probe results in a cache hit the new data and existing tag are written back to the DRAM cache. Thus, the fifth source of bandwidth bloat (referred to as *Writeback Update*) can be attributed to re-writing the tag (not data). On the other hand, if the Writeback Probe results in a cache miss there are two possibilities. If a writeback no-allocate policy is used, the data is sent to main memory. However, if a writeback allocate policy is used, the new data and new tag are written to the DRAM cache replacing the existing data. Thus, the sixth source of bandwidth bloat (referred to as *Writeback Fill*) can be attributed to updating tag and data on writeback requests.

Figure 3.3 shows the bandwidth breakdown for the Alloy cache. In a BW-Opt cache, the Bloat Factor is 1, and all the bandwidth is dedicated to Hit: The cache performs all the secondary cache operations logically, without using any of the physical resources. On the other hand, Alloy Cache requires five 128-bit bus transfers (80 bytes) to transfer the tag and data (72 bytes). This is a Bloat Factor of 1.25X for Hit compared to BW-Opt cache. Miss Probe and Miss Fill each take about 0.67X. Writeback Probe and Writeback Update each take about 0.57X. Note that DRAM cache uses a write-allocate policy, and thus it does not have Writeback Fill in the baseline. Overall, the Bloat Factor for Alloy cache is 3.8X.

Note that the cache operations corresponding to bandwidth bloat are common to both

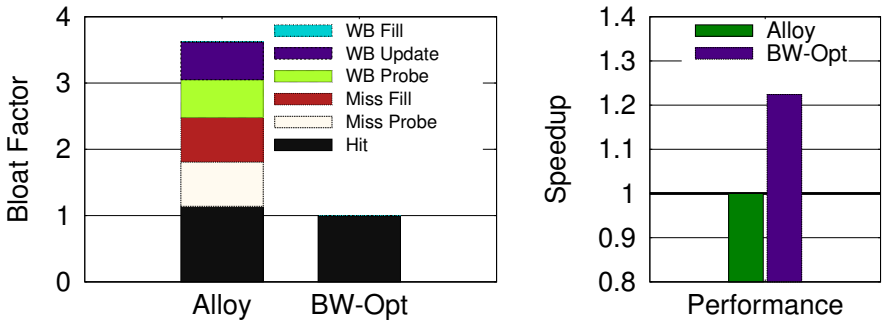

Figure 3.3: Comparison of Bloat Factor and Potential Performance.

SRAM and DRAM cache designs. However, these cache operations do not degrade performance in SRAM caches primarily because of SRAM cache implementation. Unlike DRAM caches that share a single narrow data bus for all read and write operations, SRAM caches typically consist of separate read and write ports that match the width of the corresponding tag and data. Furthermore, SRAM caches have much higher read/write bandwidth because of separate banked tag and data arrays each with their own read and write port. Therefore, the bandwidth utilized by these secondary operations has not been a critical concern for the on-chip SRAM caches. Unfortunately, for DRAM caches bandwidth is a scarce resources, so the performance overhead of these secondary operations becomes significant, and there is an opportunity to improve performance by reducing the number of cache operations that result in bandwidth bloat.

DRAM cache bandwidth bloat is attributed to six different cache operations: *Hit Probe, Miss Probe, Miss Fill, Writeback Probe, Writeback Update, and Writeback Fill.* Among these operations, only the Hit Probe contributes towards useful bandwidth to service the LLC miss request. All other cache operations are either targeted for improving performance (Miss Fill, and Writeback Fill), or for ensuring correctness (Miss Probe and Writeback Probe). As the bandwidth bloat increases DRAM cache access latency, the dissertation investigates opportunities to reduce the bandwidth bloat and proposes *Bandwidth Efficient ARchitecture (BEAR)* for DRAM caches. BEAR consists of three component techniques, each aimed at making one of the following secondary operations bandwidth efficient.

1. Bandwidth-Efficient Miss Fill. Miss Fill consumes significant DRAM cache bandwidth. A typical cache design inserts all cache lines on a miss, with the assumption that such lines will later provide cache hits. However, a significant percentage of lines are not referenced again [52, 54]. Therefore, cache bypassing is useful to reduce the bandwidth consumed by Miss Fills, even if it degrades cache hit rate by a marginal amount.

2. Bandwidth-Efficient Writeback Probe. Typically, a Writeback Probe is issued before a Writeback Update to determine whether the line already exists in the DRAM cache. If the architecture provides guarantees on whether or not a line already exists in the DRAM cache, the majority of Writeback Probes can be eliminated. The dissertation proposes enhancements to the on-chip LLC to avoid Writeback Probes.

3. Bandwidth-Efficient Miss Probe. Miss Probes waste bandwidth when the requesting line misses in the cache. The dissertation leverages DRAM cache design to buffer recently accessed neighboring tags to reduce the bandwidth of Miss Probes.

3.2 Bandwidth-efficient Miss Fill

Among secondary operations, the first target is Miss Fill, which takes 17% DRAM cache bandwidth (Bloat Factor 0.67 of 3.8). The insight is not all inserted cache lines will be re-referenced again [52, 54], which enables the opportunity that some Miss Fills bypass the DRAM cache without impacting the hit rate significantly. This section first examine a naive approach that a fixed fraction of the cache fills bypasses randomly. While such a scheme improves the cache hit latency, in some cases it causes severe degradation in both the hit rate and overall system performance. To address the issue, the dissertation proposes a *Bandwidth Aware Bypass (BAB)* scheme that frees up the bandwidth consumed by Miss Fills while limiting the degradation of hit rate to a predetermined amount.

3.2.1 Probabilistic Bypass: A Simple and Naive Scheme

A fairly simple and straight forward way to reduce the bandwidth consumed by Miss Fill is to not perform Miss Fill for a given percentage of cache misses. Let the *Bypass Probability* (P) denote the fraction of total cache misses for which skip the Miss Fill and instead bypass the cache. On a cache miss, the decision whether to install of not can be made by consulting a random number generator. If the value of the random number generator is less than P, perform bypass; otherwise fill the line in the cache. This scheme is referred to as *Probabilistic Bypass (PB)*. The parameter P regulates the effectiveness of PB at reducing the bandwidth consumed by Miss Fills. At high value of P, a larger number of lines bypasses the cache, which reduces the bandwidth consumption of Miss Fills, and therefore improves the cache hit latency. Unfortunately, bypassing a larger number of cache lines can have adverse impact on the hit rate too, and thus harm overall system performance. To analyze this phenomenon, two values of bypass probability are studies: P=50% and P=90%. Note, PB with P=0% is the same as the baseline design which does not perform bypass.

Figure 3.4 shows the reduction in cache hit latency (higher is better), increase in cache hit rate (higher is better) and speedup (higher is better) with PB for P=50% and P=90%. As expected, aggressive bypassing reduces hit latency significantly, on average by 12% for P=90%. Unfortunately, probabilistic bypassing also decreases the hit rate significantly for several workloads (such as *Gems* and *zeusmp*), which degrades performance. Overall, the speedup of probabilistic bypass is negligible, and PB is ineffective at improving performance.

3.2.2 Bandwidth Aware Bypass: Limiting Hit-Rate Loss

Ideally, while the benefits from the reduction of the cache hit latency is desirable, the cache hit rate should not be penalized significantly. For the DRAM cache to be high performing, PB should not degrade the cache hit rate significantly with respect to the baseline. This suggests a dynamic mechanism that measures the differential in hit rate (or miss rate) between

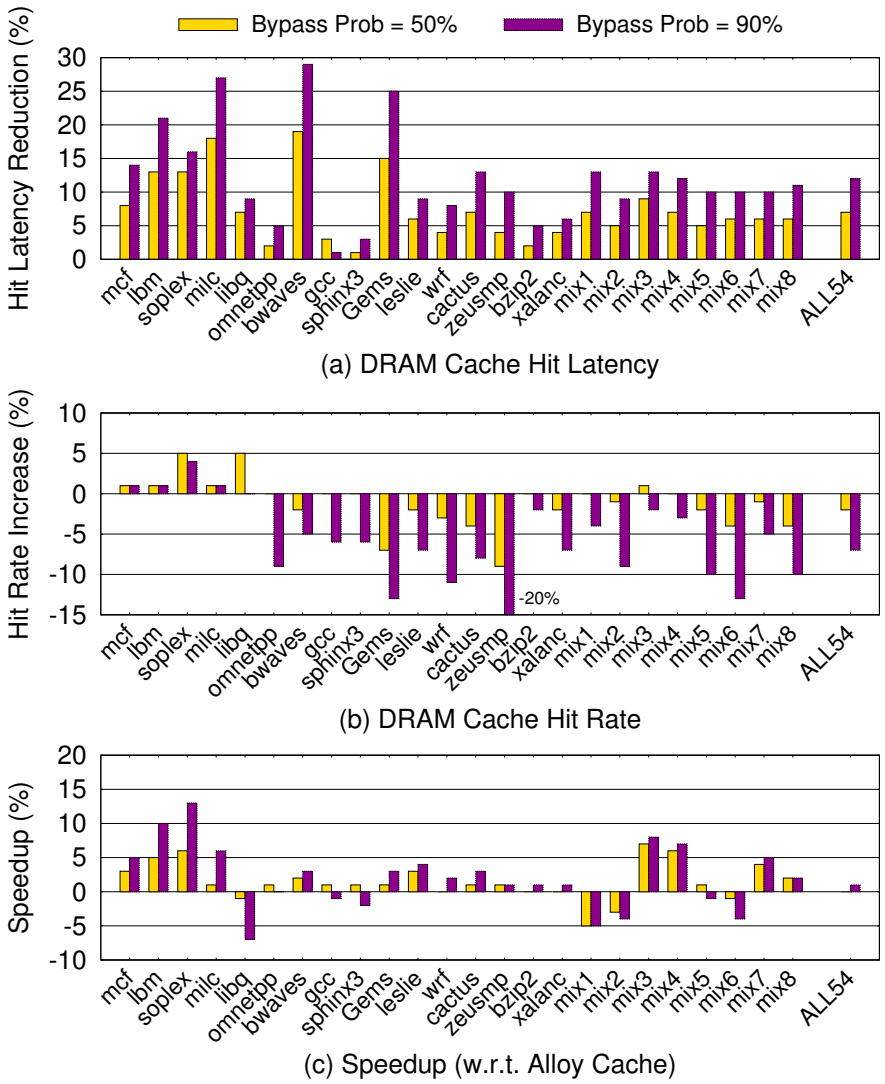

Figure 3.4: Comparison of Probabilistic Bypass with P=50% and P=90% in terms of impact on (a) Cache Hit Latency (b) Cache Hit Rate (c) Speedup. All numbers are with respect to the baseline.

the baseline and PB. If the differential is lower than some threshold then it should use PB, otherwise the baseline. This mechanism is referred to as *bandwidth aware bypass (BAB)*, as it tries to continue to bypass (to free up the bandwidth) even if such bypassing causes a minor degradation in cache hit rate. This is unlike prior schemes on cache replacement that aim to do bypassing solely with the aim of maximizing cache hit rate, and would try to disable the bypassing mechanism if there is any loss of hit rate.

The overview of BAB is shown in Figure 3.5. BAB uses set dueling [52] to dynamically select between PB and the baseline. Of the 16M sets in the DRAM cache, BAB creates two sampling monitors of 512K sets each for PB and the baseline policy, and the remaining 15M sets are the follower sets. BAB uses two 16-bit counters for each sampling monitor: one counts misses, and the other counts accesses. Misses and accesses to the sampled sets increment the corresponding 16-bit counters. When any of the access counters saturates, all the counters are shifted right by 1 bit. The miss rates of the baseline and PB are calculated for sampled sets and then the difference of the two miss rates is compared to a threshold, Δ. If the difference is smaller than the threshold, PB and baseline have similar miss rates. Therefore, BABs sets the mode bit to enable the follower sets to use PB. Whereas, if the difference is greater than or equal to the threshold, the baseline has better miss rate, and BAB unsets the mode bit to enable the follower sets to use the baseline policy. Note that there is a single mode bit for the entire cache and it changes only when one of the access counter saturates.

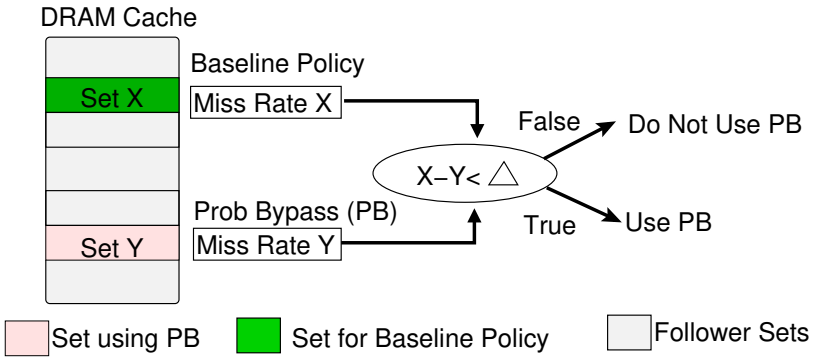

Figure 3.5: Design of Bandwidth Aware Bypass

3.2.3 Effectiveness of BAB

A sensitivity study using 90% probability to determine the best threshold for the differential in the miss rate for the mechanism to select between PB and the baseline shows that using $\Delta = \frac{1}{16}$ gave the best overall performance, which means that PB must provide a hit rate of at least 15/16th of the baseline hit rate for the bypassing to continue, otherwise PB get disabled. Figure 3.6 shows the speedup of BAB, in which the component PB policy uses a bypass probability of 90%. On average, BAB improves the performance by 5.1% (and as much as 15%) over the baseline, without causing degradation in any of the workloads. The cache hit rate with and without BAB are 61% and 63%, respectively. Thus, BAB sacrifices a small amount of cache hit rate to free the Miss Fill bandwidth, which reduces the hit latency and improves the performance.

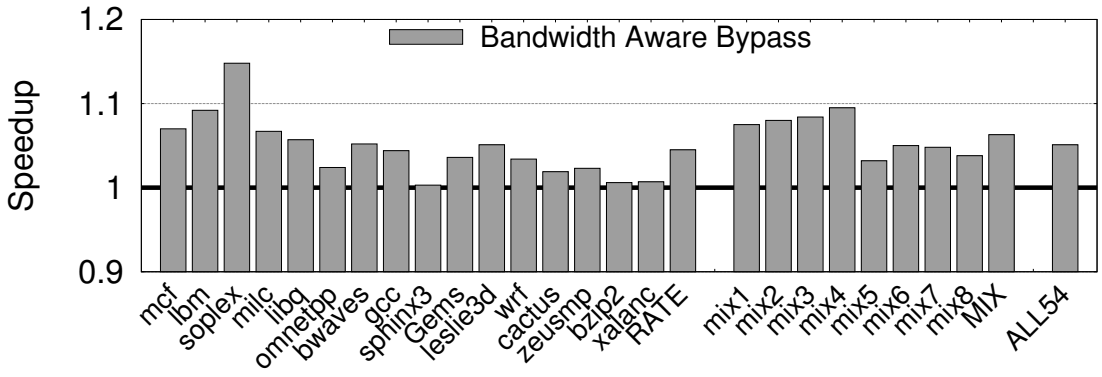

Figure 3.6: Speedup from Bandwidth Aware Bypass

3.3 Bandwidth-Efficient Writeback Probe

Cache writebacks update main memory with new data modified by the processor core. A writeback request from LLC must consult the DRAM cache via a WriteBack Probe to determine whether the line already exists in the DRAM cache. The probe is necessary for correctness to make sure that the DRAM cache services future requests with the most recent data value. In general, a Writeback Probe is wasteful if the line evicted from the

on-chip LLC (i.e., dirty line) already exists in the DRAM cache. As DRAM caches are generally much larger than on-chip LLCs, the probability that a writeback request misses in the DRAM cache tends to be very low (¡ 1% in this study). This suggests that the majority of Writeback Probes are useless and cause unnecesary bandwidth bloat. Hence, it is highly desirable if the cache architecture can provide some guarantees on whether (or not) a dirty line evicted from the on-chip LLC exists in the DRAM cache.

3.3.1 Limitation of Inclusive Caches

One approach is to enforce the inclusion property [63] for the DRAM cache. Enforcing inclusion mandates that all lines resident in the small on-chip caches must also be resident in the DRAM cache. When evicting lines from the DRAM cache, inclusion is enforced by sending a back-invalidate request to also evict the line from all on-chip caches (should the line be present). Therefore, inclusion property for DRAM caches eliminates the need for Writeback Probes as writebacks are guaranteed to hit in the DRAM cache. However, inclusion prevents bypassing of cache lines on misses and eliminates the 5-15% performance benefits from the bandwidth conscious bandwidth-aware bypass policy. Therefore, the dissertation investigates a desirable mechanism that reduces not only Writeback Probes but also Miss Fills.

3.3.2 Tracking Residency of Line in DRAM Cache

Writeback Probes can be avoided if there exists some state information in the cache hierarchy that specifies which cache lines are resident in the DRAM cache. Note that this state information need not be for every line in the DRAM cache, but only those lines that are dirty in the on-chip caches. Therefore, the state information can be reduced from tracking millions of lines in the DRAM cache to only a few thousand lines present in the on-chip caches. The state information is a one-bit field that tracks whether or not a line is present in the DRAM cache. This one-bit field is referred to as *DRAM cache presence (DCP)*, which

is one extra bit in the tags of each line in the LLC, shown in Figure 3.7. DCP is modified on LLC fills and DRAM cache evictions. On LLC fills, DCP is set to one if the line is serviced from the DRAM cache and zero, otherwise. DCP is kept up-to-date on DRAM cache evictions. When a line is evicted from the DRAM cache, the LLC is conveyed this information (similar to the flow of an inclusive DRAM cache). If the line is present in the LLC, DCP is updated to zero (instead of invalidating the line as in inclusive cache). Therefore, the LLC knows that the line is no longer present in the DRAM cache.

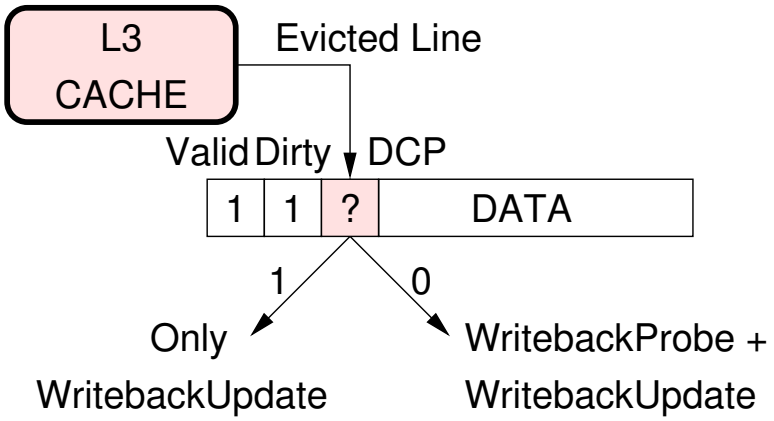

Figure 3.7: Design of DRAM Cache Presence Bit

DCP enables writeback requests to have full knowledge on whether or not the line is present in the DRAM cache. On writeback requests from the LLC, if the DCP value is one, meaning the line is in the DRAM cache, only a Writeback Update is necessary to update the content in the DRAM cache. which avoids the bandwidth bloat of Writeback Probes. On the other hand, a DCP value of zero implies a writeback miss as the line is no longer present in the DRAM cache. To ensure correctness, a Writeback Probe is necessary to determine whether the writeback request replaces a dirty line from the DRAM cache (assuming writeback-miss allocate policy). If the resident line is dirty, the DRAM cache must write back the dirty line to the memory.

3.3.3 Effectiveness of DRAM Cache Presence

Figure 3.8 illustrates the performance improvement of DCP in the presence of BAB. DCP improves performance by an additional 4%, with maximum of 12.8% in *omnetpp*, and 11.3% in *gcc*, both of which have very high hit rate for writebacks.

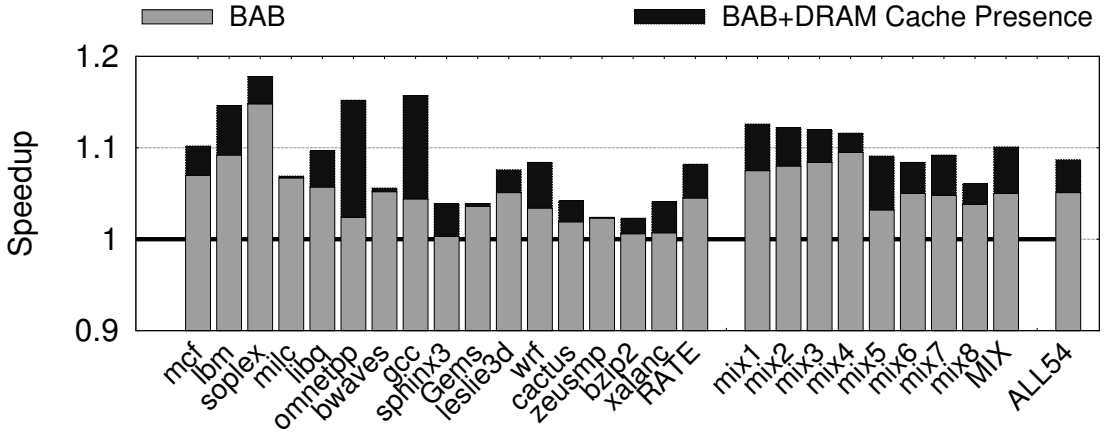

Figure 3.8: Performance for DRAM Cache Presence (DCP) over the baseline system that implements Bandwidth-Aware Bypass.

3.4 Bandwidth-Efficient Miss Probe

DRAM cache look-ups result in either a cache hit or a cache miss. Cache hits result in useful bandwidth whereas cache misses unnecessarily waste bandwidth by needlessly fetching *clean* data that are not utilized. Such bandwidth bloat is referred to as Miss Probe. If the DRAM cache architecture provides some guarantees on whether (or not) a line is present in the DRAM cache, Miss Probe bandwidth bloat can be minimized. The insight is that current DRAM cache designs, including both Loh-Hill and alloy cache, locate tag and data together in the same DRAM row buffer; thus, accessing one cache line also reads tags of other adjacent lines, making additional information available. The alloy cache organizes the tag and data together to form a single tag and data (TAD) entry. Each TAD entry is 72 bytes long (8 bytes for tag and 64 bytes for the data). Alloy cache organizes consecutive cache sets into the same row buffer as illustrated in Figure 3.9. With a 128-bit (16-byte)

3D-DRAM data bus, a cache lookup transfers a TAD entry in five bursts (a total of 80 bytes are transferred).

3.4.1 Neighboring Tag Cache

In alloy cache, any cache lookup also transfers the neighboring tag of the line present in the next cache set. This spatial locality can be exploited by storing the neighboring tag in a small fully associative structure, referred to as the *neighboring tag cache (NTC)*. Each NTC entry contains two fields: tag and DRAM cache set index. A miss in the LLC first consults the NTC by performing a set index match and tag match. If there is no set index match, the NTC can not provide any guarantees on the existence of the line in the DRAM cache. Therefore, a Miss Probe must be issued to to determine DRAM cache hit or miss. If there is a set index match and a tag match, the NTC guarantees that the request is present in the DRAM cache. Finally, if there is a set index match but a tag mismatch, the NTC gurantees that the line is not present in the DRAM cache. In this situation, the NTC can reduce the bandwidth bloat of Miss Probes. Note that if the tag suggests that the DRAM cache entry is dirty, a Miss Probe is still necessary for correctness to read the dirty line and write it back to main memory.

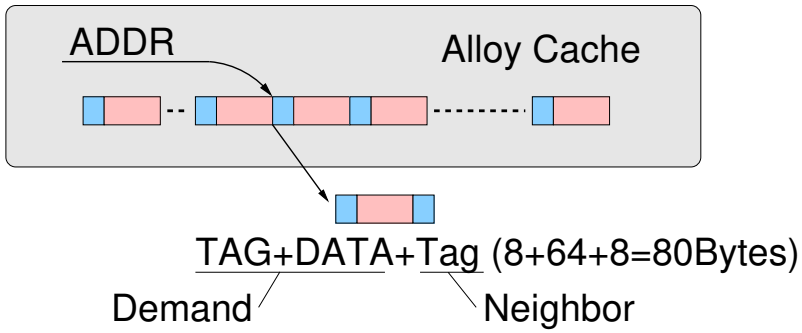

Figure 3.9: Alloy Cache brings in two tag entries with each access by default (due to bus being 16 bytes and tag being 8 bytes).

3.4.2 Effectiveness of NTC

NTC has 8 entries for every DRAM bank. For a DRAM cache with four channels and 16 banks per channel, the overall NTC size is 512 entries. However, only the eight NTC entries that correspond to the DRAM cache bank are accessed on an LLC miss, which incurs one cycle access latency. The NTC is kept up-to-date on DRAM cache evictions. Figure 3.10 shows that NTC improves performance by an additional 2%. Detailed analysis reveals that the NTC provides performance benefits by reducing two sources of wasteful bandwidth. First, by design NTC reduces the bandwidth bloat of Miss Probes. Second, as a side benefit, the DRAM cache miss predictor takes advantage of the NTC to verify parallel memory access predictions. If a given entry is present in the NTC, the DRAM cache miss predictor squashes the wasteful parallel access to main memory.

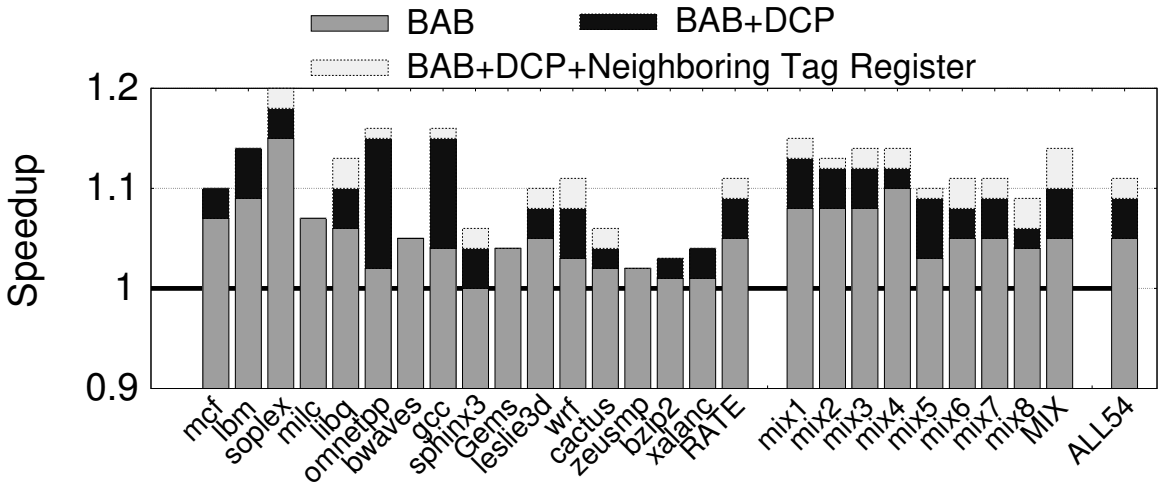

Figure 3.10: Performance for Neighboring Tag Cache for a baseline with BAB and DCP.

3.5 Methodology

3.5.1 System Configuration

The experiments are conducted on a x86 simulator with a detailed memory system model [64]. Table 3.1 shows the configuration used in our study. The cache-memory system has a four-

level cache hierarchy (L1, L2, L3 being on-chip SRAM caches and L4 being the off-chip DRAM cache). All cache hierarchy uses 64B line size. The baseline L4 cache is alloy cache, and the results are normalized to alloy cache unless stated otherwise. Cache misses fill all levels of the hierarchy. The alloy cache is equipped with a the MAP-I miss predictor [30] to overcome the tag look-up latency for cache misses. The baseline assumes that L4 is non-inclusive of L3 cache. (L3 cache can be either inclusive or non-inclusive of L1/L2 caches, although this dissertation models non-inclusive L3 cache for simplicity). A virtual memory system performs virtual to physical address translations.

Table 3.1: Baseline System Configuration for Bandwidth-Efficient DRAM Cache Study

Processors	
Number of Cores	8
Frequency	3.2GHz
Core Width	2 wide out-of-order
Last Level Cache	
Shared L3 Cache	8MB, 16-way, 24 cycles
DRAM Cache	
Capacity	1GB
Bus Frequency	1.6GHz (DDR 3.2GHz)
Channels	4
Banks	16 Banks per rank
Bus Width	128 bits per channel
tCAS-tRCD-tRP-tRAS	36-36-36-144 CPU cycles
Main Memory (Conventional DRAM)	
Capacity	16GB
Bus Frequency	800MHz (DDR 1.6GHz)
Channels	2
Banks	8 Banks per rank
Bus Width	64 bits per channel
tCAS-tRCD-tRP-tRAS	36-36-36-144 CPU cycles

The heterogeneous memory system is based on HBM technology [9] and commodity DDR-based DRAM technology [4]. In accordance with the specification for 3D-DRAM, the latency is same as that of commodity DRAM. However, the bandwidth of DRAM cache is much higher than main memory. In the baseline, DRAM cache has 8x bandwidth as main memory (2X channel, 2X bus width, 2X bus frequency). The DRAM timing is based on

USIMM [64]. For both the 3D-DRAM and commodity DRAM, each memory channel has separate read queue and write queue, the scheduler prioritizes read requests over write requests, and writes are issued in batches.

3.5.2 Workloads

The workloads are selected based on a representative region of 1-billion instructions from the SPEC CPU2006 benchmark suite [65], captured by SimPoints [66]. Only workloads have Miss Per Thousand Instruction (MPKI) greater than 1 are studies, shown in Table 3.2. The workloads are grouped into two categories: High Intensive (MPKI greater than 12) and Medium Intensive (MPKI between 2 and 12). The experiments executes benchmarks in rate mode, where all eight cores execute the same benchmark. Also, 38 mixed workloads that are selected from the above 16 benchmarks are evaluated. The virtual-to-physical page mapping ensures that two benchmarks do not map to the same address.

Table 3.2: Workload Characteristics for Bandwidth-Efficient DRAM Cache Study

Category	Name	L3 MPKI	Footprint
High Intensive	mcf	74.6	10.2 GB
	lbm	32.7	3.1 GB
	soplex	27.1	1.9 GB
	milc	26.1	4.5 GB
	libquantum	25.5	256 MB
	omnetpp	21.1	1.1 GB
	bwaves	18.7	1.5 GB
	gcc	18.6	680 MB
Medium Intensive	sphinx3	12.4	136 MB
	GemsFDTD	9.9	5.3 GB
	leslie3d	7.6	616 MB
	wrf	6.8	488 MB
	cactusADM	5.5	1.2 GB
	zeusmp	4.8	1.5 GB
	bzip2	3.7	2.4 GB
	xalancbmk	2.3	1.3 GB

3.5.3 Figure of Merit: Performance and Bandwidth

For rate mode workloads, the performance reports the total execution time. The reported speedup is the normalized execution time with respect to the baseline system. For mixed workloads, the performance metric uses weighted speedup, and the reported speedup is the normalized weighted speedup with respect to the baseline system. The average performance uses geometric mean that shows the average speedup for the 16 rate-mode runs (RATE), 8 mix-mode runs (MIX), and all 54 workloads (ALL). In addition, another important aspect is the bandwidth consumption of the DRAM cache. The metric, *bloat factor*, is defined as the amount of total bytes transferred divided by the useful bytes transferred on the bus, as shown in Equation 3.1. The denominator also means the total cache lines transferred to the processor multiplied by cache line size (i.e., 64 bytes). Note that bandwidth efficiency is the inverse of the Bloat Factor.

3.6 Results and Analysis

3.6.1 Overhead of BEAR

Table 3.3 show the hardware overhead of each proposal. Overall, BEAR incurs negligible hardware overhead of 19.2 KB, the majority of which is due to the DCP-bit in the LLC.

Table 3.3: Storage Overhead of BEAR

Design	Cost
Bandwidth-Aware Bypass	8 bytes per thread, total 64 bytes
DRAM Cache Presence	One bit per line in LLC, total 16K bytes
Neighboring Tag Register	44 bytes per bank, total 3.2K bytes
Total	19.2K bytes.

3.6.2 Overall Performance

BEAR consists of three schemes to reduce bandwidth bloat in the DRAM cache. Figure 3.11 shows the performance of between the baseline alloy cache, BEAR, and an ideal

bandwidth optimized (BW-Optimized) DRAM cache. Note that RATE and MIX are referred to as the geometric mean of rate and mix workloads, while ALL54 is the geometric mean of all of our 54 workloads. On average, BEAR outperforms the alloy cache by 10.1%, while BEAR outperforms the BW-Optimized DRAM cache in some workloads: *soplex*, *milc*, and *libq* because bandwidth-aware bypass increases the hit rate for these benchmarks, which reduces overall memory latency and thus provides better performance than BW-Optimized cache. This is not the typical case, however, as BAB causes a hit rate degradation of 2%, on average. Table 3.4 shows the hit rate and the latency of DRAM cache. On average, BEAR reduces the DRAM cache hit latency from 239 to 182 cycles, which is a 24% improvement, while only sacrificing a 2% hit rate. In addition, the miss latency reduces because the side effect of neighboring tag cache reduces unnecessary parallel access to the off-chip memory.

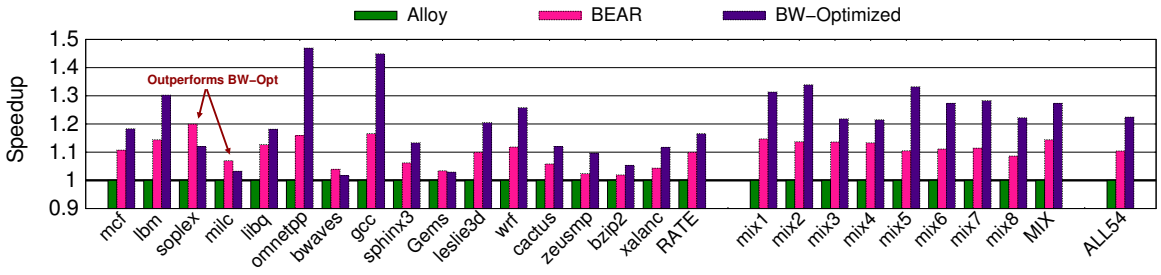

Figure 3.11: Performance Improvement for Alloy, BEAR, and ideal case. Note that RATE and MIX are for 16 rate mode workloads, and 8 mixed workloads, respectively; ALL54 means the geometric mean across all 54 workloads.

Table 3.4: Comparison of DRAM Cache Hit-Rate and Latency.

Design	Hit Rate	Latency (cycles)		
		Hit	Miss	AVG
Alloy	63.2%	239	391	326
BEAR	**61.0%**	**182**	356	282

3.6.3 Impact of Bloat Factor

Figure 3.12 illustrates the effectiveness of our proposals in reducing bandwidth bloat by illustrating a bandwidth breakdown for every bandwidth factor, including Hit, Miss Probe, Miss Fill, Writeback Probe, Writeback Update, and Writeback Fill normalized to the Bloat Factor of a BW-Optimized DRAM cache. The BW-Optimized case only consumes Hit bandwidth, and transfers 64 bytes for every request. For other configurations, the basic unit of data transfer is 80 bytes. In the baseline Alloy cache, the Bloat Factor on average is 3.8, in which only 1.25 is critical to service the LLC miss requests. To review, bandwidth-aware bypass (BAB) targets reduction in Miss Fills, dRAM cache presence (DCP) targets reduction in Write Probes, and neighboring tag cache (NTc) targets reduction in Miss Probe. Overall, BEAR is able to reduce the Bloat Factor by 32%.

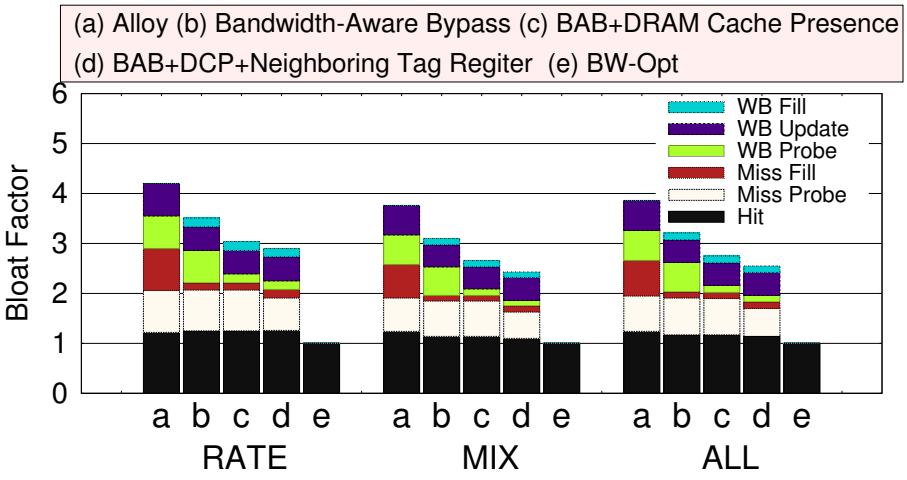

Figure 3.12: Bloat Factor for Different Schemes.

3.6.4 Sensitivity to DRAM Cache Bandwidth and Capacity

In the default system parameter, the bandwidth of DRAM cache is 8X as that of the off-chip DRAM. This section studies the scalability and the sensitivity by varying the DRAM caches bandwidth from 4X, 8X to 16X (by varying the number of channels) while keeping the cache size constant. Figure 3.13(a) shows the performance improvement when the

DRAM cache bandwidth varies. BEAR continues to provide performance improvements of more than 10% over the baseline alloy cache for all the bandwidth configurations. Besides the bandwidth sensitivity study, Figure 3.13(b) also shows the sensitivity study by varying the size of the DRAM cache while keeping the bandwidth constant. The range of the cache size changes from 512MB to 2GB. BEAR consistently improves performance by more than 10% across all DRAM cache capacity. Therefore, BEAR is scalable regardless of the bandwidth or the capacity provided by DRAM caches.

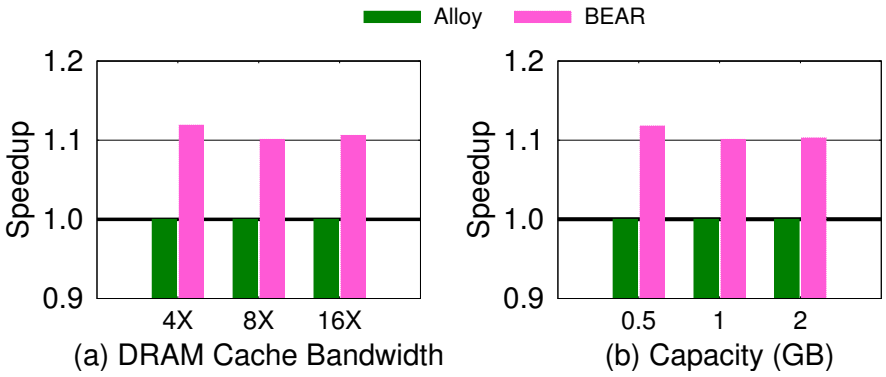

Figure 3.13: Sensitivity to DRAM Cache: (a) Bandwidth (b) Capacity. Note that all numbers are normalized to Alloy cache with respect to each configuration.

3.6.5 Comparison to Alternative Tags-in-DRAM Designs

The comparison to various implementations of DRAM cache, including Loh-Hill cache (LH-cache) [21], mostly-clean cache (MC-Cache) [67], and inclusive alloy cache (Incl-Alloy) [30], is discussed in this section. The LH-cache assumes a MissMap structure that has the same latency of LLC, which is 24 cycles. MC-cache, an extension of LH-cache, reduces the Miss Probe bandwidth and issues memory requests to off-chipe memory. MC-cache assumes a perfect predictor for hits and misses, and if the outcome of the predictor is a miss, the request will be serviced by the off-chip memory. For Incl-Alloy cache, the DRAM cache is inclusive with respect to the on-chip LLC. Figure 3.14 shows the performance comparison, in which the baseline is a system without DRAM caches. LH-cache has 27% performance improvement across all the workloads, while MC-cache has 30%. Incl-

Alloy cache improves performance by 55% on average, which is 9% more than baseline non-inclusive alloy cache. Nevertheless, BEAR provides 66% performance improvement.

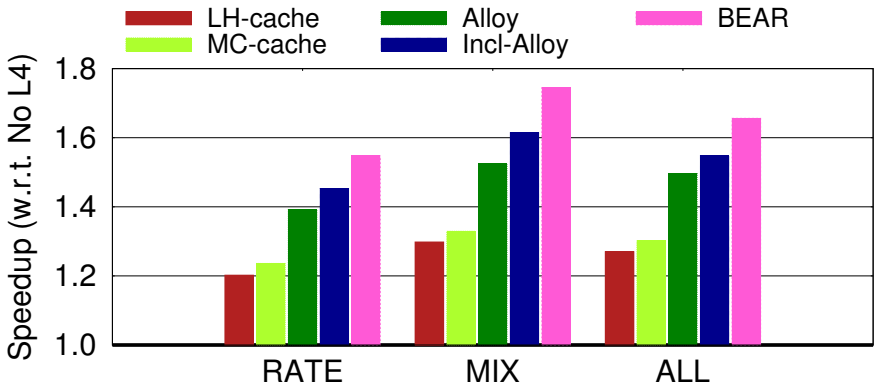

Figure 3.14: Speedup from different implementations of DRAM cache, normalized to a system without DRAM caches.

3.6.6 Analysis of Tags-In-SRAM Designs

Besides tags-in-DRAM designs, this section analyzes tags-in-SRAM designs. An unconstrained version of such a design, referred to as *Tags In SRAM (TIS)* stores all the tags on-chip in an SRAM structure and incurs a prohibitive storage of 64MB (at four byte of tag storage per line). The SRAM storage can be reduced to 6MB by architecting the cache as a sector cache (SC) [23], which has been considered in recent designs such as the *footprint cache* [24]. The advantage of these SRAM-based designs is that they can support high set associativity and avoid some probe operations (e.g. Miss Probe and Writeback Probe). Unfortunately, these advantages come at a high storage overhead and also high latency overheads of accessing the tag store, before accessing the data store. The analysis assumes that both TIS and SC have appropriate SRAM structure for tag store without penalizing either design for the added storage or the extra latency of tag access. Both caches are architected as 32-way set associative. SC uses 4KB as an sector, which has 64 blocks (64B). The statistics related to the DRAM cache (L4), including L4 hit rate, L4 hit latency, L4 miss latency, and Bloat Factor as well as the speedup are shown in Figure 3.15.

BEAR outperforms tags-in-SRAM cache designs for the following reasons. (1) **Hit Rate:** For a gigascale DRAM cache, the set associativity contributes only to a limited improvement in hit rate (from 63% to 68%, consistent with prior studies [30].) (2) **Bloat Factor:** Both TIS and SC still incur bandwidth bloat from Miss Fill, Writeback Update, and Dirty Evictions. BEAR has very similar Bloat Factor as TIS and SC, because the amount BEAR saves is close to the amount of Miss Probe TIS and SC save. One can use the principles of BEAR to reduce the bandwidth Bloat of TIS and SC also. (c) **Latency:** The latency is the decisive reason for the performance difference. Although SRAM caches do not need to look up tags in DRAM to detect cache misses, they do incur the penalty of dirty replacement, which gets exacerbated in SC as an evicted page can have a large number of dirty lines. Overall, BEAR has 10.1% performance improvement, which exceeds the 7.5% speedup with TIS and 18% slowdown with SC. BEAR requires an SRAM overhead of only 20KB, whereas TIS and SC incur respective 64MB and 6MB SRAM storage overhead, which may be prohibitive.

Figure 3.15: Comparison to Tags-In-SRAM (TIS) Cache and Sector Cache (SC): (a) L4 Hit Rate, (b) L4 Hit Latency, (c) L4 Miss Latency, (d) Bloat Factor, and (e) Speedup (w.r.t. Alloy). Note that TIS requires 64MB SRAM storage and SC requires 6MB SRAM storage.

3.7 Summary

This chapter discusses the bandwidth efficiency of DRAM caches. Maintaining the cache functionality, DRAM caches require six DRAM cache operations: Hit Probe, Miss Probe, Miss Fill, Writeback Probe, and Writeback update, and Writeback Fill. Among those op-

erations, only Hit Probe contributes to satisfy the miss request from L3 cache, and secondary bandwidth factor are either for performance or for correctness. A metric, referred to as *bloat factor*, is defined to measure how these secondary operations use DRAM cache bandwidth: Only 33% of the DRAM cache bandwidth is used to satisfy L3 miss requests. Other secondary operations use the DRAM cache bandwidth, increase the queuing delay as well as the DRAM cache hit latency. If bandwidth bloats are eliminated, DRAM cache hits can be serviced quickly.

To mitigate the bandwidth bottleneck in DRAM cache, this dissertation proposes Bandwidth-Efficient ARchitecture (BEAR) for DRAM cache. BEAR has three different schemes, each of which targets its own bandwidth component: bandwidth aware bypass for Miss Fill, DRAM cache presence for Writeback Probe, and neighboring tag cache for Miss Probe. Overall, the three component schemes of BEAR can be implemented with a storage overhead of only 4KB (and one bit per line in the L3 cache). BEAR can be implemented without any changes to the architecture of the DRAM array. The evaluations show that BEAR reduces the bandwidth consumption of DRAM cache by 32% and improves system performance by 10.1%. BEAR achieves half the performance possible from an idealized bandwidth-optimized design that consumes no bandwidth for any of the secondary operations. Besides the tags-in-DRAM designs, the bandwidth bloat is a problem for Tags-in-SRAM designs, too. BEAR outperforms an idealized design that stores the tags on-chip using 64MB SRAM and the sector cache design that incurs an SRAM overhead of 6MB.

CHAPTER 4

MOSAIC

This chapter examines the problem in the case that the DRAM cache is used with non-volatile memory in heterogeneous memory systems. Non-volatile memory, such as *phase change memory (PCM)* offers 4-8X capacity as large as and also better scalibilty in sub-20nm process than the commodity DRAM [68].

4.1 Problem: Is Set Associativity Useful?

In addition to a configuration that a future heterogeneous memory system has 3D-DRAM and commodity DRAM, a future heterogeneous memory system has 3D-DRAM for high bandwidth and PCM for high capacity. One key difference for PCM-based heterogeneous memory systems is that accesses to PCM are slower: read latency of PCM is twice as that of commodity DRAM, while write latency is four times. The long read latency of PCM is on the critical path of a DRAM cache miss; therefore, the key priority to architect DRAM cache for PCM-based heterogeneous memory systems is to avoid the DRAM cache miss penalty by improving the DRAM cache hit rate. As the current DRAM cache is organized as a direct-mapped cache, one simple way to improve the cache hit rate is to architect the DRAM cache as a set-associative cache. However, although set associativity improves the DRAM cache hit rate and reduces the chance of miss penalty, it incurs the overhead of checking multiple ways by performing multiple 3D-DRAM accesses. The overhead may compensate the benefits and jeopardize the use of set associativity.

To understand the trade-offs of hit-rate improvement and the overhead of extra bandwidth consumption, the dissertation conducts a study that compares the performance of a set-associative DRAM cache to that of a direct-mapped DRAM cache. Similar to the configuration of the recent commercial product [31], the direct-mapped DRAM cache places

tags in the 3D-DRAM and uses 64B cache-line size. The set-associative DRAM cache uses a two-way set-associative DRAM cache, in which two ways in the same set are placed adjacently in the same 3D-DRAM row buffer. The tags are also placed in the 3D-DRAM and associated with the lines. In this case, a DRAM cache request reading tags and data of both ways (as prior studies suggest [26, 30], the two-way cache reads both ways by pipelining read commands, instead of sequentially checking one way and then the other). Therefore, for every DRAM cache request, the two-way DRAM cache incurs twice bandwidth assumption as the direct-mapped DRAM cache, and the extra bandwidth consumption increases the DRAM-cache access latency.

Figure 4.1(a) shows the performance of both direct-mapped and two-way DRAM cache (detailed methodology in Section 4.3). The baseline is the direct-mapped cache (labeled as *DM*). On average, the two-way DRAM cache outperforms the direct-mapped cache by 9%. Although some benchmarks such as *soplex, libq, and zeusmp* show descent performance improvement, the two-way cache significantly degrades performance of other benchmarks such as *mcf, omnetpp, and sphinx3* as much as 20%. Figure 4.1(b) further investigates the cause of performance improvement or degradation and show the DRAM cache hit rate and hit latency. On average, the two-way DRAM cache improve the cache hit rate by 6.3% but increases the cache hit latency by 31%. For workloads with performance improvement, the hit-rate improvement is large enough (5% or more), which compensates the hit-latency overhead. Whereas, for workloads with performance degradation the hit-rate improvement is little, which cannot mitigate the hit-latency overhead.

Because of its high bandwidth consumption, the two-way cache always increases the DRAM-cache hit latency, but the two-way DRAM cache does *not* always improve the hit rate. Ideally, a system should pay the overhead only when the set associativity is useful and avoid the overhead when it is not. That is, when the set associativity does not improve the hit rate, the DRAM cache should be organized as a direct-mapped cache to avoid the penalty of increased hit latency; otherwise, the DRAM cache should be organized as a two-

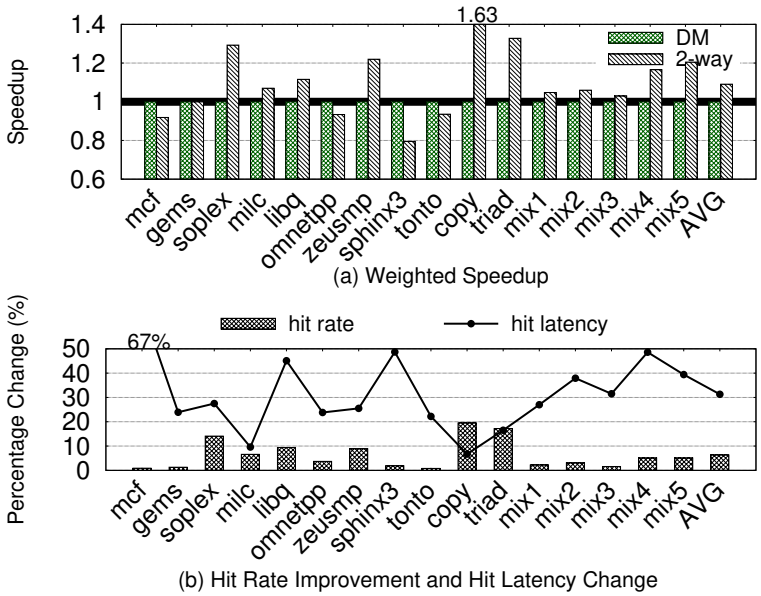

Figure 4.1: Comparison of a direct-mapped DRAM cache (*DM*) and a two-way DRAM cache. (a) Performance based on normalized weighted speedup and (b) Hit rate improvement and hit latency change of a two-way DRAM cache.

way cache to capitalize the improvement of the hit rate and the reduction of miss penalty. To this end, this dissertation proposes *morphable set-associative DRAM cache (MOSAIC)* that dynamically changes its organization based on the hit rate improvement provided by the set associativity.

4.2 Morphable Set-Associative DRAM Cache

Although switching between a direct-mapped cache and a two-way cache addresses the issues of set associativity for DRAM caches, it is not trivial to architect the DRAM cache to have such capability. This section examines a naive way that converts the whole cache from one to another, which incurs a huge overhead. Figure 4.2(a) illustrates the direct-mapped cache and its mapping to the 3D-DRAM, which places contiguous cache sets adjacently in one 3D-DRAM row buffer [30]. Also, Figure 4.2(b) shows a two-way cache that places both cache ways of the same cache set adjacently and contiguous cache sets adjacently in one 3D-DRAM row buffer. The figures assumes that the cache has 8 cache lines and show

possible memory addresses that each set contains. In the direct-mapped case, set 0 hosts address 0 (A0), followed by set 1 that contains address 1 (A1); in the two-way case, set 0 hosts address 0 or 4 (A0 or A4); set 1 address 1 or 5 (A1 or A5). When the set associativity changes, the location of lines must comply with the sets. That is, A1 should be in set 1 regardless of the current set associativity; set 1, however, is at the second location in a direct-mapped cache but at the third location in a two-way cache, so A1 must change its location or be invalidated. Thus, for correctness, either direction of transitions must check *all* lines and rearrange their locations if necessary. After the checking, the *whole* cache starts using the *same* desired set associativity.

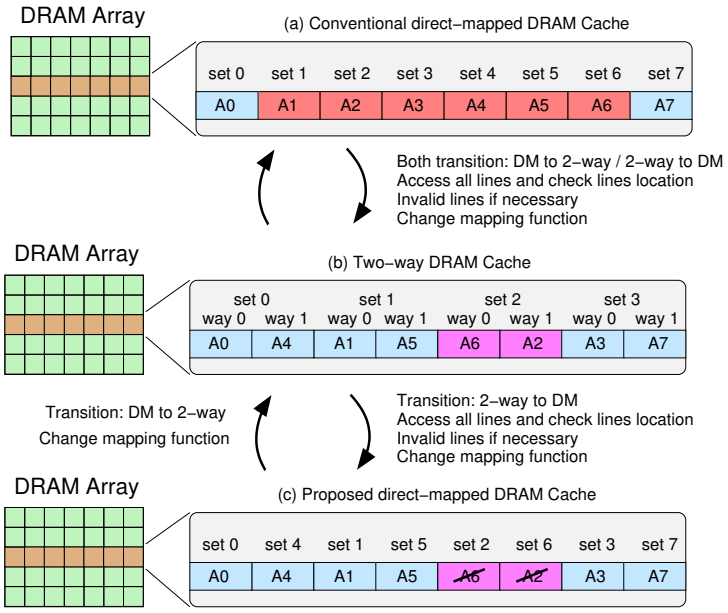

Figure 4.2: Design of Morphable Set-Associative DRAM Cache: (a) a conventional direct-mapped cache, (b) a two-way set-associative cache, and (c) the proposed set mapping function for the direct-mapped cache. The transition overhead from direct-mapped to two-way significantly reduces.

4.2.1 Low-Cost Transition Between A Direct-mapped Cache and A Two-way Cache

The transitions in the naive way incurs a huge overhead by reading (and likely invalidating) all lines in the cache. Avoiding the need to access the whole cache and thus reducing the number of accesses mitigates such overhead during a transition. To this end, this disser-

tation proposes an infrastructure that allows a local decision of the set associativity: part of the cache is direct-mapped and other part two-way, and each individual part changes its set associativity independently. As the transition is localized, only the part of cache that is accessed transitions . Figure 4.2(c) presents a direct-mapped DRAM-cache set-mapping scheme such that the locations of all addresses in such a mapping are also eligible locations in the two-way cache. The mapping scheme interleaves contiguous sets with sets that could have contained addresses in the location if the cache were a two-way cache: Set 4 in the location between Set 0 and Set 1, Set 5 between Set 1 and Set 2, and so on. In this mapping, address 4 (A4) belongs to Set 4, second location to the left, which is an eligible location of A4 in the two-way cache. Notice that one set in a two-way cache represents two sets in a direct-mapped cache and these two lines form a basic unit that decides its set associativity.

As the infrastructure allows the cache to be partially direct-mapped and partially two-way, a cache request must know the information of the set associativity. To amortize the storage cost of such information and also exploit spatial locality, MOSAIC groups multiple units that use the same set associativity and associates one bit per group for the information. The bit is referred to as *morphable set associativity bit*. In this study, MOSAIC groups 64 contiguous two-way sets (128 cache lines) as one group, referred to as *morphable set-associative group (MoSAG)*, shown in Figure 4.3. For a 2GB DRAM cache, the storage overhead is 32KB, less than 0.4% of the area of on-chip caches. Applying the set associativity to only one MoSAG, significantly reduces the transition overhead. The overhead incurred by the transition from the direct-mapped set associtivity to the two-way one includes only the change of the morphable set-associative bit. On the other hand, the transition from two-way to direct-mapped, unfortunately, must check all lines' location. For example, in Figure 4.2(b), set 2 has two addresses of A3 and A6. These two lines are not in the eligible location if the cache were direct-mapped; therefore, they must be invalidated before the cache becomes direct-mapped, shown in Figure 4.2(c). However, unlike the naive scheme, the check in MOSAIC is limited within the MoSAG, which avoids the

need to access the whole cache.

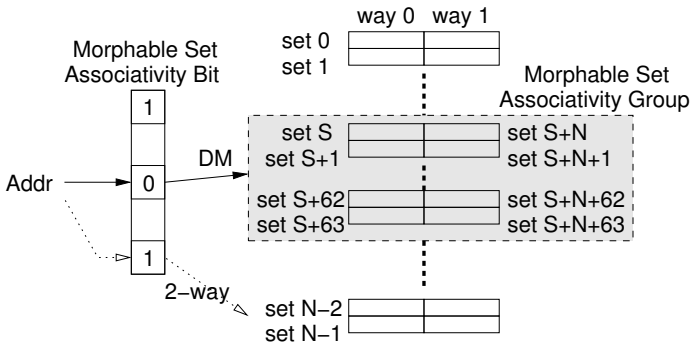

Figure 4.3: Design of morphable set associativity group. MOSAIC uses 64 two-way sets as a group that employs the same set associativity. A cache request first checks the corresponding bit of the group before using the indexing scheme of the designated set associativity to access the location.

4.2.2 Morphing Between Set Associativities

With the low-cost infrastructure, MOSAIC still needs a mechanism that decides the appropriate set associativity and also enforces such decision. Described in Section 4.1, the two-way cache *sometimes* improves the cache hit rate but *always* increases the cache hit latency. Therefore, the cache should use the two-way cache only if the two-way cache yields a significant improvement of the hit rate over the direct-mapped cache; otherwise, if the improvement of the cache hit rate is small, the direct-mapped cache. To estimate the hit rate, MOSAIC employs group sampling, similar to set sampling [52, 69]. MOSAIC dedicate a small number of the MoSAG that is always either direct-mapped or two-way, shown in Figure 4.4. The dedicated MoSAGs are referred to as *leader MoSAG* and other MoSAGs are *follower MoSAG*. It calculates the cache hit rate by using 16-bit counters and evaluate the delta that is the cache hit rate of the two-way MoSAG minus that of the direct-mapped MoSAG. If the delta is greater than a threshold, MOSAIC sets the set associativity as two-way. If the delta is smaller than zero (meaning direct-mapped is better), MOSAIC uses a direct-mapped cache. The decision is referred to as *global set associativity (GSA)*.

After the set associativity is determined, the cache needs to enforce the decision. Unlike the naive mechanism in Figure 4.2(a), MOSAIC do not enforce the decision by converting the whole cache immediately after the decision is made. Instead, MOSAIC use a *lazy* transition such that the enforcement occurs only on a cache miss. When the missed cache line returns from the memory, MOSAIC examines if the set associativity of the MoSAG matches the GSA determined by the leader MoSAGs. A mismatch triggers the transition process that converts the MoSAG to the GSA. After the transition finishes, the cache starts using the GSA. The benefits of the lazy conversion are two fold: First, it does not stall the system for a long time converting the whole cache immediately after the GSA changes. Second, it reduces the transition overhead because only MoSAGs that have misses are converted. That is, although the GSA and the local set associativity disagree, if the follower MoSAG has no misses, MOSAIC does not convert it to a GSA MoSAG. The frequency of the transition depends on the threshold of the hit rate difference. The dissertation performs a sensitivity study that varies the threshold from 0.5% to 12%, which shows that 3% as the threshold performs the best. For the rest of the chapter, the default threshold is 3%.

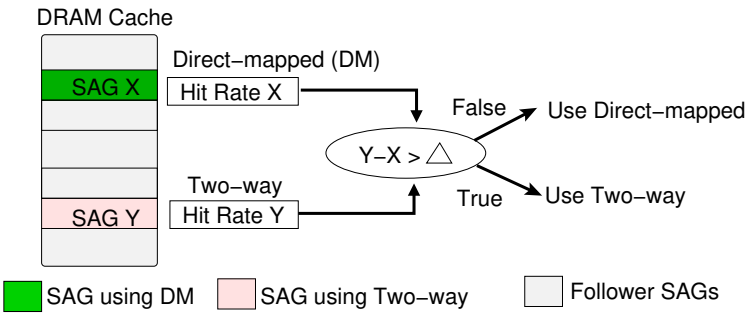

Figure 4.4: Design of Hit-Rate-Driven Morphable Set-Associative Cache (MOSAIC)

4.3 Experimental Methodology

The experiments are condcuted on a x86 simulator with a detailed memory system model, USIMM[64]. Table 4.1 shows the configuration used in the study. The system has a four-level cache hierarchy (L1, L2, L3 being on-chip SRAM caches and L4 being the off-chip

DRAM cache). All cache hierarchy uses 64B line size. The baseline is a direct-mapped DRAM cache, similar to recent commercial products [31]. The two-way cache uses a random replacement policy to avoid the status update overhead incurred by LRU policy. The DRAM cache is based on HBM technology [9], while the main memory PCM technology. Each memory channel has separate read queue and write queue, and the scheduler prioritizes read requests over write requests and issues writes in batches.

Table 4.1: System Configuration for PCM-based Heterogeneous Memory Systems

Processors	8 cores: 3.2GHz, 2-wide OoO core
Shared L3 Cache	8MB, 16-way, 24 cycles
DRAM Cache	
Capacity	2GB
Bus Frequency	1000MHz (DDR 2GHz)
Configuration	4 channel, 64-bit bus
tCAS-tRCD-tRP-tRAS	36-36-36-144 CPU cycles
Main Memory (PCM)	
Capacity	64GB
Bus Frequency	1000MHz (DDR 2GHz)
Configuration	1 channel, 64-bit bus
tCAS-tRCD-tRP-tRAS	36-36-36-144 CPU cycles

The workloads are selected from a representative region of 2-billion instructions from the SPEC CPU2006 [65] and STREAM [70] benchmark suite. The experiments executes benchmarks in rate mode, where all eight cores run the same benchmark. The study also evaluates 54 mixed workloads that are selected from high misses-per-kilo-instruction (MPKI) benchmarks. The virtual-to-physical page mapping ensures that two benchmarks do not map to the same address. Because of the space constraint, only select benchmarks of the total 73 workloads are shown in the analysis but all workloads are shown in Section 4.4.3. The performance metric is weighted speedup and the reported speedup is normalized to the baseline system. Also, the average speedup is a geometric mean of all 73 workloads (labeled *ALL*).

4.4 Results and Analysis

4.4.1 The Effectiveness of MOSAIC

MOSAIC allows each MoSAG to independently be either direct-mapped or two-way. The effectiveness of MOSAIC depends on if MOSAIC chooses the right set associtivity. Figure 4.5 shows the distribution of cache requests that access either a direct-mapped MoSAG or a two-way MoSAG. Some workloads, such as *mcf, sphinx3, and tonto*, largely remains direct-mapped, while other workloads present a mix of direct-mapped and two-way accesses. The reason of the mixed accesses is because MOSAIC converts the follower MoSAG only on DRAM cache misses; therefore, if a follower MoSAG that is direct-mapped does not have any misses, it would not be converted to a two-way MoSAG, and cache requests to the MoSAG uses the direct-mapped set associativity.

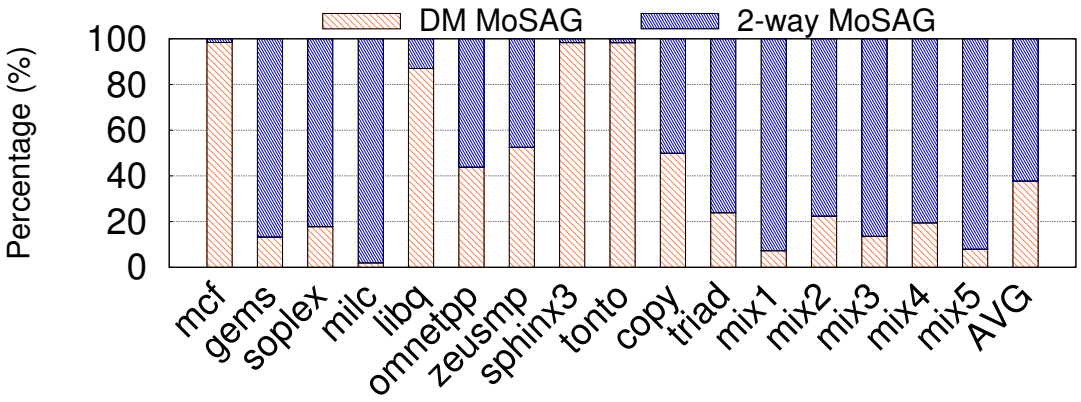

Figure 4.5: The distribution of accesses to a direct-mapped or two-way MoSAG in MOSAIC.

4.4.2 Hit Rate and Hit Latency

The objective of MOSAIC is to mitigate the hit latency overhead of a two-way cache but at the same time retains the benefits of the hit rate improvement. Figure 4.6 shows the change of both the hit rate and the hit latency for the two-way cache and morphable set-associative cache (labeled *MOSAIC*). For workloads that does not benefit from the two-way cache, such

as *mcf, sphinx3, and tonto*, MOSAIC is able to use the direct-mapped cache and avoid the hit latency overhead. For workloads that show a significant hit rate improvement, such as *soplex, libq, and copy*, MOSAIC uses the two-way scheme and improve the hit rate. Also, as a result of the lazy transition, many cache requests in MOSAIC access direct-mapped MoSAGs, which reduces the bandwidth consumption and thus the hit latency overhead of a two-way cache.

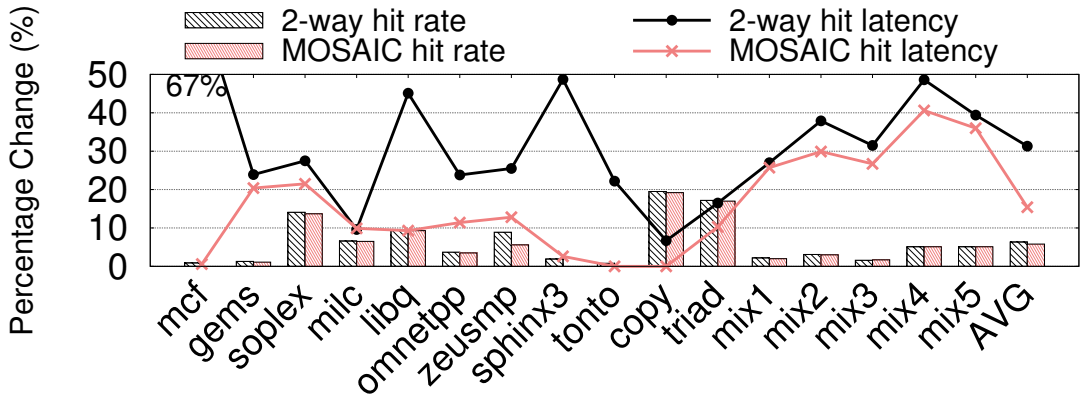

Figure 4.6: Hit rate improvement and hit latency change of two-way and MOSAIC.

4.4.3 Performance of All Workloads

The performance of all 73 workloads are shown in Figure 4.7. The workloads are sorted based on the performance of a two-way cache, normalized to the baseline direct-mapped cache and shown in s-curve. Although the two-way cache outperforms the direct-mapped cache by an average of 9.2%, it degrades performance for several workloads as much as 20%. MOSAIC mitigates most the degradation, with a maximum degradation of 1.8% and outperforms the baseline by an average of 10.3%.

4.5 Summary

For heterogeneous memory systems that have high-bandwidth 3D-DRAM as a cache and high-capacity PCM as the memory, a key priority is to avoid the DRAM cache miss penalty

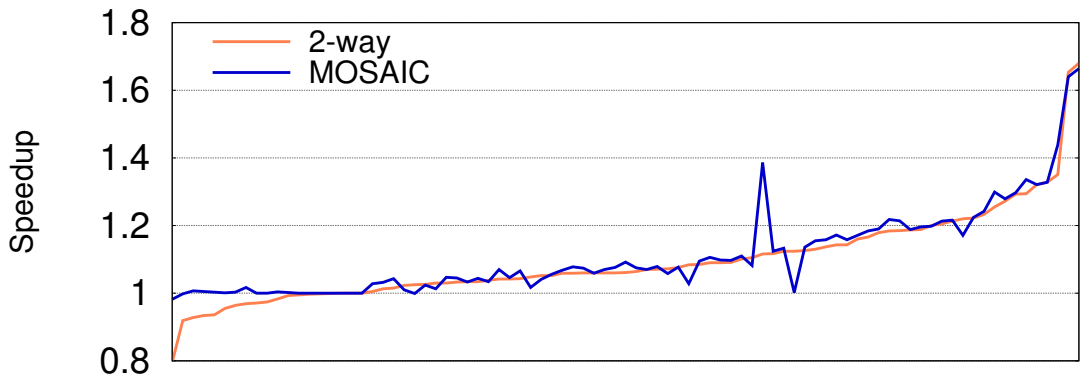

Figure 4.7: S-curve performance of all workloads for two-way and MOSAIC.

that accesses long-latency PCM. Although recent studies propose a direct-mapped DRAM cache to optimize the cache hit latency, it suffers from higher miss rate than a two-way cache. However, while a two-way cache sometimes improves the cache hit rate, it always increases the cache hit latency; therefore, a two-way cache degrades the performance of some workloads by as much as 20%. Ideally, the system should pay the overhead only when the set associativity is useful. To this end, the dissertation proposes a morphable set-associative cache (MOSAIC), which is composed of two parts: an infrastructure that allows the cache to transition between two levels of set associativities with low overhead and a mechanism that dynamically chooses the best-performance set associativity depending on the hit-rate improvement. MOSAIC mitigates the overhead of a two-way cache but keeps the benefits of the hit-rate improvement. The experiments show that while a two-way cache has a maximum of performance degradation of 20%, MOSAIC limits such degradation to less than 1.8%.

CHAPTER 5
MAXIMIZING THE SYSTEM-BANDWIDTH UTILIZATION OF HETEROGENEOUS MEMORY SYSTEMS

This chapter investigates the problem of system-bandwidth utilization of heterogeneous memory systems. 3D-DRAM offers four to eight times bandwidth as commodity DRAM, so the aggregate system bandwidth is five times: four times from 3D-DRAM and one time from commodity DRAM. However, the conventional wisdom tends to use only the bandwidth of 3D-DRAM, leaving 20% of the system bandwidth unused. This chapter analyzes the optimal bandwidth utilization and proposes a simple mechanism that achieves such memory access distribution. This chapter refer to commodity DRAM as *Comm-DRAM* and interchangeably uses heterogeneous memory systems with *tiered-memory systems*.

5.1 Problem: Sub-optimal System-bandwidth Utilization

5.1.1 Conventional Wisdom: Optimize for Hit Rate

Figure 5.1(a) shows a typical heterogeneous memory system available in commercial products today: a 4x-bandwidth 3D-DRAM and a 1x-bandwidth commodity DRAM, referred to as *Comm-DRAM*. The 3D-DRAM can be architected in two ways: a *cache mode* or a *flat mode* (two-level memory). Regardless of how the 3D-DRAM is architected in a heterogeneous memory system, the conventional wisdom maximizes the fraction of memory requests satisfied by the 3D-DRAM (i.e., the access rate of the 3D-DRAM) [17, 26, 29]. Doing so inefficiently utilizes total available bandwidth in the system. For example, Figure 5.1(b) presents an application whose frequently accessed working set fits in the 3D-DRAM and shows a system that employs the conventional approach: On an access to data in the Comm-DRAM, the conventional approach always moves the data to the 3D-DRAM,

shown in Figure 5.1(b), so later accesses to the data are serviced by the 3D-DRAM. In a steady state, the entire working set is always serviced by the 3D-DRAM. As a result, the utilized system bandwidth is 4x (only 3D-DRAM), as shown in Figure 5.1(c).

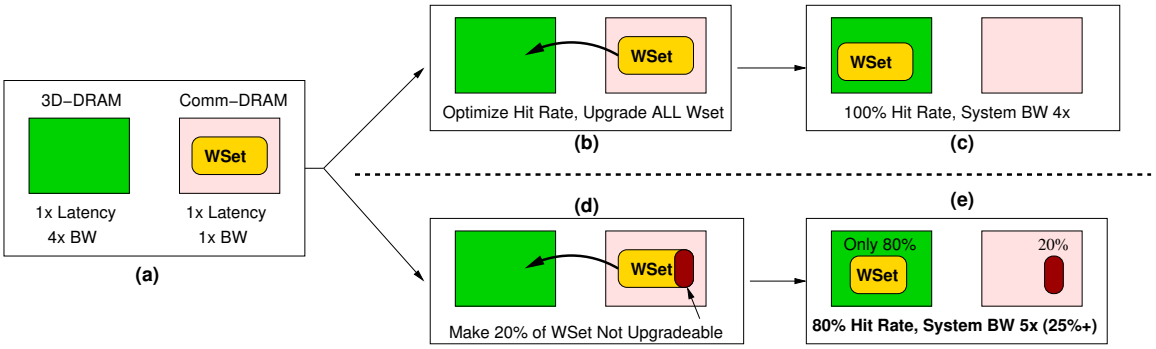

Figure 5.1: Optimizing for the access rate versus for system bandwidth. (a) a system with 3D-DRAM and commodity DRAM (Comm-DRAM), both having the same latency but 3D-DRAM has 4x the bandwidth, (b) and (c) traditional systems that try to optimize for hit rate gives up to 4x system bandwidth, (d) and (e) explicitly controlling some part of the working set to remain in Comm-DRAM results in up to 5x system bandwidth.

$$\text{The 3D-DRAM Access Rate} = \frac{\text{Total 3D-DRAM Accesses}}{\text{Total Memory Accesses}} \tag{5.1}$$

As 3D-DRAM simply provides higher bandwidth, not lower latency, placing all the frequently accessed data in the 3D-DRAM does not necessarily yield optimal performance. Unlike the conventional designs, which evaluates its effectiveness according to the 3D-DRAM access rate, the key metric for a heterogeneous memory system is the number of memory requests satisfied within a time period, or, simply put, the overall system bandwidth. Higher overall system bandwidth leads to better system performance. In a conventional approach that optimizes for the access rate of 3D-DRAM, the overall system bandwidth is under-utilized and capped by the bandwidth of the 3D-DRAM. Ideally, to maximize overall system bandwidth and performance, all of the bandwidth available from both the 3D-DRAM and the Comm-DRAM should contribute.

5.1.2 Optimize for the Overall System Bandwidth

The heterogeneous memory system utilizes the bandwidth of both the 3D-DRAM and the Comm-DRAM by distributing memory accesses to both DRAM. This section corroborates this hypothesis experimentally: As an example, this section studies a set of memory intensive STREAM benchmarks [70] on a memory system whose configuration is similar to that in Figure 5.1 (4x bandwidth 3D-DRAM and 1x bandwidth Comm-DRAM, see the experimental methodology in Section 5.4). As the working set of the STREAM benchmarks fits in the 3D-DRAM, the baseline configuration services all memory accesses entirely from the 3D-DRAM (a 100% 3D-DRAM access rate). A sensitivity study distributes memory accesses to both DRAM by explicitly allocating part of the working set in the Comm-DRAM. Figure 5.2 shows the speedup compared to the baseline as the the fraction of memory requests serviced by the 3D-DRAM increases from 10% to 100%. For all systems that vary the bandwidth ratio from 2X, 4X, to 8X, the peak performance occurs much earlier than 100%, validating the hypothesis that the system optimizing for overall system bandwidth achieves higher performance.

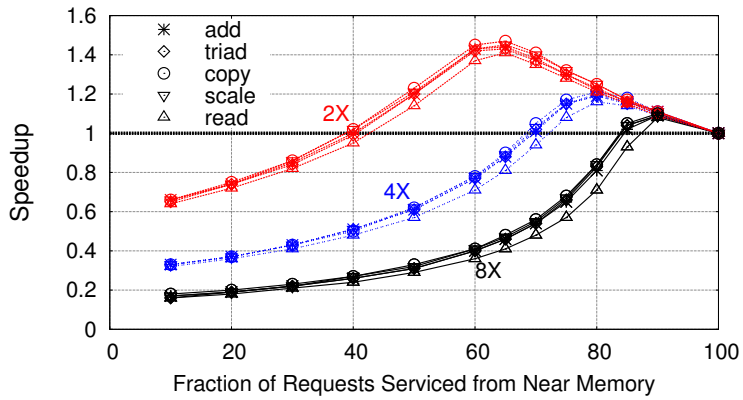

Figure 5.2: Speedup versus access rate of 3D-DRAM. Note the peak performance occurs much earlier than 100% (baseline system).

The maximum performance improvement depends on the overall bandwidth improvement. When the ratio of the bandwidth of the 3D-DRAM to that of the Comm-DRAM is 4x, the performance peaks at 1.2x, close to the ideal speedup of 1.25x, which is ex-

pected by improving the system bandwidth by 25% (system bandwidth increases from 4x to 5x, or 25% improvement). A similar observation applies to systems with different bandwidth ratios; systems with 2x and 8X bandwidth ratio have the peak performance of 1.45x and 1.11x, respectively, while the bandwidth improves by 50% and 12.5%, respectively. Therefore, for a heterogeneous memory system in which the Comm-DRAM accounts for a significant fraction of the overall bandwidth, distributing memory accesses to both DRAM has great potential to improve performance.

5.1.3 Goal: Optimum Split at Runtime

Peak performance occurs when memory accesses are distributed in proportion to the bandwidth of each DRAM. When the 3D-DRAM has 4x as high bandwidth, peak performance occurs when the access rate of the 3D-DRAM is approximately 80%: The 3D-DRAM services four-fifths of the accesses and the Comm-DRAM the remaining one-fifth of the accesses. The same principle applies to the case of 2X and 8X the bandwidth ratio. This observation that maximum system bandwidth and peak performance occur when memory accesses are distributed in proportion to the respective bandwidth is consistent with recent studies [71, 72], one of which is a study by Agarwal et al. on heterogeneous memory systems for GPU [71]. The authors propose a static page placement strategy that relies on programmers knowledge of the data structures in workloads. However, the study has several drawbacks: First, their scheme requires software modification. Second, the proposed static scheme cannot adjust the memory access distribution at runtime. As the prior work has limited applicability and requires programmer and software intervention, this dissertation seeks a mechanism that achieves the desired memory access distribution at runtime without any software support.

The key insight that achieves the desired access distribution is the explicit control of data movement in heterogeneous memory systems. When data moves from the Comm-DRAM to the 3D-DRAM, subsequent memory access to the data will be serviced by the

3D-DRAM, which increases the 3D-DRAM access rate. Examples of such data movement include cache line install in the cache mode and page migration in the flat mode. Therefore, controlling data movement can regulate the 3D-DRAM access rate. For instance, in Figure 5.1(d), when the 3D-DRAM already has 80% of memory accesses, if further data movement from the Comm-DRAM to the 3D-DRAM is disallowed and Comm-DRAM keeps data that account for 20% of memory accesses, the system achieves the desired split of memory accesses that maximizes the system bandwidth utilization. In such case, the overall system bandwidth has the maximum of 5x (3D-DRAM + Comm-DRAM), which is 25% higher than that of the conventional approach, shown in Figure 5.1(e).

5.2 BATMAN: Bandwidth-aware Management

To explicitly regulate the access distribution, this dissertation proposes *bandwidth-aware tiered-memory management (BATMAN)*, a runtime mechanism that manages the memory accesses distribution in proportion to the bandwidth ratio of the 3D-DRAM and the Comm-DRAM. The desired 3D-DRAM access rate is referred to as the *target access rate* (TAR). BATMAN monitors the access rate of the 3D-DRAM at runtime and controls the data movement to meet the TAR: When the 3D-DRAM access rate exceeds the TAR, BATMAN disallows data movement from the Comm-DRAM to the 3D-DRAM and proactively moves data from the 3D-DRAM to the Comm-DRAM, which lowers the 3D-DRAM access rate. Also, when the 3D-DRAM access rate falls below the TAR, BATMAN does not intervene in data movement that increases the 3D-DRAM access rate. BATMAN demonstrates its effectiveness in the context of two use cases This section examines systems in the cache mode and develops the fundamental mechanism of BATMAN, while Section 5.3 presents BATMAN in the flat mode

5.2.1 Idea: Controlling the 3D-DRAM Access Rate by Partially Disabling the Cache

The mechanism of BATMAN that controls the 3D-DRAM access rate becomes the regulation of the cache hit rate for DRAM caches. Although other cache operations, such as miss-related and writeback-related operations, also contribute to the 3D-DRAM accesses [69], the cache hit rate is a proxy of the 3D-DRAM access rate in the cache mode. For example, when the DRAM cache has a 100% hit rate, all memory requests are serviced by the 3D-DRAM (DRAM cache), which leads to a 100% 3D-DRAM access rate. To achieve the goal of regulating the 3D-DRAM access rate at TAR, BATMAN monitors the 3D-DRAM access rate at runtime and takes action based on either of the following two cases: (1) an 3D-DRAM access rate higher than the TAR or (2) an 3D-DRAM access rate lower than the TAR. In the first case, BATMAN intentionally lowers the cache hit rate to achieve the TAR. On the other hand, in the second case, BATMAN uses the baseline mechanism, which increases the cache hit rate. Therefore, in either case, BATMAN forces the system to approach the TAR.

One simple way of dynamically regulating the DRAM cache hit rate is via partial cache disabling. When a cache set is disabled, memory accesses to the disabled set would miss in the DRAM cache and use the Comm-DRAM to obtain data. In an extreme case in which all the cache sets are disabled, all memory accesses would miss in the cache and rely on the Comm-DRAM for data, which results in both a 0% cache hit rate and a 0% 3D-DRAM access rate. Therefore, explicitly controlling the number of disabled sets is the key to controlling the 3D-DRAM access rate. When the 3D-DRAM access rate is higher than the TAR, BATMAN disables more cache sets, which lowers the rate. On the other hand, when the 3D-DRAM access rate is lower than the TAR, BATMAN enables the disabled sets, which increases the 3D-DRAM access rate. To convert an 3D-DRAM access to an Comm-DRAM access, memory accesses to disabled cache sets should not incur an 3D-DRAM access. However, to maintain data integrity between the DRAM cache and the memory, a memory access to a disabled set must check the DRAM cache (a tag

lookup via an 3D-DRAM access), ensuring that the most recent copy of requested data is not in the cache, which consumes 3D-DRAM bandwidth. To conserve the bandwidth of such accesses, BATMAN supports the disabling of DRAM cache sets by pre-selecting a subset of the DRAM cache sets as candidates that can be disabled. With the knowledge of pre-selected sets, BATMAN avoids the tag look-up overheads (3D-DRAM accesses) when memory requests go to disabled sets.

5.2.2 Design of BATMAN for DRAM Caches

Figure 5.3(a) presents an overview of BATMAN for DRAM caches. The key attribute that controls the number of disabled cache sets is the *disabled sets index (DSIndex)*. The pre-selected sets at an index lower than DSIndex are the "disabled sets," which neither incur a tag look-up overhead nor service any cache requests, shown in Figure 5.3(b). The pre-selected sets at an index higher than DSIndex are enabled sets that can still service cache requests. By moving the DSIndex to different positions, BATMAN controls a fraction of cache sets that remain enabled and thus the 3D-DRAM access rate. BATMAN regulates the movement of DSIndex by monitoring the 3D-DRAM access rate and comparing it to the target access rate (TAR, 80% in the default parameters). The system monitors the 3D-DRAM access rate and to increase or decrease the DSIndex, as described below:

Structures

BATMAN monitors the access rate of the 3D-DRAM using two 16-bit counters: *Access-CounterCache* and *AccessCounterTotal*. While the AccessCounterCache counter tracks the total number of the 3D-DRAM accesses, including reads, tag look-ups and writebacks, the AccessCounterTotal counter tracks the total number of accesses to the 3D-DRAM and to the 3D-DRAM. The 3D-DRAM access rate is the value of the AccessCounterCache counter divided by the AccessCounterTotal counter. When the AccessCounterTotal counter overflows, both counters halve (right shift by one) their value. Figure 5.3(a) shows the tracking

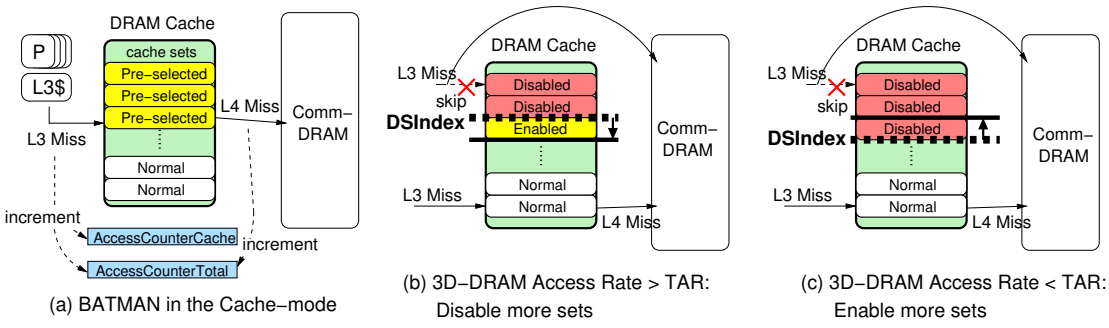

Figure 5.3: Overview of BATMAN for DRAM caches. (a) BATMAN in the cache mode. BATMAN uses two counters to monitor the 3D-DRAM access rate: *AccessCounterCache* and *AccessCounterTotal*. While one access to the 3D-DRAM increments both counters, one access to the Comm-DRAM increments only the AccessCounterTotal counter. BATMAN selects cache sets as candidates that can be disabled. (b) All pre-selected sets at index lower than DSIndex are "disabled sets," which neither incur a tag look-up nor service any cache request. When the 3D-DRAM access rate is greater than the TAR, DSIndex increases and disables more cache sets. (c) When the 3D-DRAM access rate is lower than the TAR, DSIndex decreases and enables the disabled sets.

counters and their operations. BATMAN requires only two 16-bit counters and a 32-bit DSIndex, which has a negligible storage overhead of eight bytes.

Operation

Memory requests are serviced by either the 3D-DRAM or the Comm-DRAM based on the status of cache set that they access in the DRAM cache. If a L3 miss goes to a disabled set, a pre-selected set whose index is smaller than the DSIndex, (e.g., the top request in Figure 5.3(b)), the request directly goes to the Comm-DRAM, without the need for a cache look-up. On the other hand, if a L3 miss goes to other sets, either normal sets or pre-selected sets whose index is larger than the DSIndex (e.g., the bottom request in Figure 5.3(b)), the request follows a normal operation: It looks up the cache set to find the corresponding data block. If the request hits in the cache, the request is serviced by the 3D-DRAM; otherwise, the request goes to the Comm-DRAM for data and installs the data in the cache.

Regulating DSIndex and Hysteresis

BATMAN continuously monitors the 3D-DRAM access rate by computing the ratio of AccessCounterCache to AccessCounterTotal to determine whether the 3D-DRAM access rate exceeds the TAR. If the 3D-DRAM access rate exceeds the TAR, BATMAN increases the DSIndex until the rate is within a 2% guard-band of the TAR. If the 3D-DRAM access rate is lower than the TAR, BATMAN decreases DSIndex until the rate is within the 2% guard-band. For a fast converge time, the length of each DSIndex movement, either an increase or a decrease, is proportional to the delta between the measure 3D-DRAM access rate and the TAR.[1] Before increasing the DSIndex, BATMAN flushes the sets that would be disabled because the index of the sets becomes lower than the DSIndex. Note that the cache flushing overhead includes reading the data from the DRAM cache and writing the data back to the memory should the block is dirty.

5.2.3 The 3D-DRAM Access Rate with BATMAN

As the goal of BATMAN is to regulate the 3D-DRAM access rate, the 3D-DRAM access rate of workloads over time during the execution is a key metric: The 3D-DRAM access rate recorded every 20 million cycles are shown for three sampled workloads in Figure 5.4. In each one, the x-axis is the execution time normalized to the baseline, the y-axis is the 3D-DRAM access rate, and two configurations, the baseline and BATMAN, are shown in the graph. Figure 5.4(a) and Figure 5.4(b) are *copy* and *soplex*, which have an almost 100% 3D-DRAM access rate in the baseline; BATMAN is effective at regulating the 3D-DRAM access rate at 80%, the TAR. In addition, Figure 5.4(c) is *lbm*, with different phases that have various access rates. BATMAN is effective at adapting to the phases and maintaining the 3D-DRAM access rate at the TAR.

[1]The figure discusses the design of contiguity in disabled sets only for simplicity. In the implementation, BATMAN chooses the candidates (i.e., pre-selected sets) based on hashed indexing. That is, only every *Nth* set is eligible for cache disabling; also, the DSIndex changes by N on a movement. For the default parameters, N is five. The advantage of hashed indexing is the reduction of the traversal time, in which the DSIndex reaches to hot regions of sets that are far away from the DSIndex, by N times.

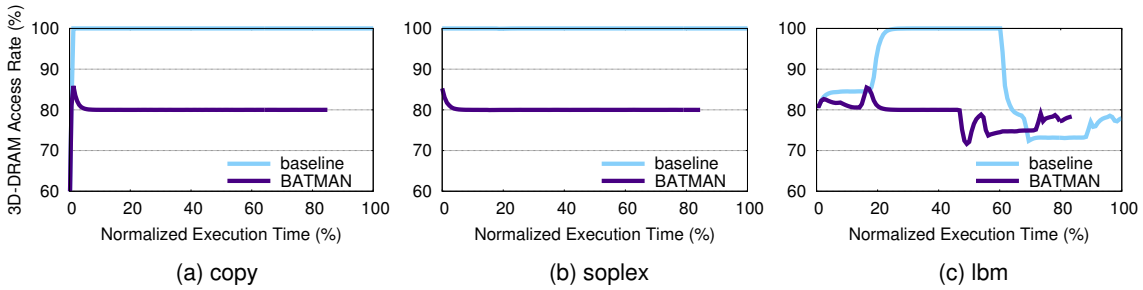

Figure 5.4: 3D-DRAM Access rate versus time. In each figure, the x-axis is the execution time normalized to the baseline, and the y-axis is the 3D-DRAM access rate recorded per 20-million-cycle interval. The 3D-DRAM access rate of both the baseline and BATMAN for three workloads: (a) copy, (b) soplex, and (c) lbm.

In addition to the 3D-DRAM access rate per interval, Figure 5.5 presents the average 3D-DRAM access rate for both the baseline system and BATMAN. Recall that the baseline system has no disabled cache sets and always installs cache lines in the cache after a cache miss. In the baseline, the access rate of the 3D-DRAM is 95% on average, with many workloads exceeding 90%. Workloads with a high 3D-DRAM access rate is consistent with the reported workloads of Intel Knights Landing [31]. These workloads have a working set size that is smaller than the 3D-DRAM capacity, so their working set fits in the 3D-DRAM, which results in a high 3D-DRAM access rate. In contrast, for workloads whose working set size is larger than the 3D-DRAM (SPEC_BIG), the 3D-DRAM access rate is not as high. For those workloads whose 3D-DRAM access rate is over the TAR, BATMAN consistently regulates the rate at the TAR, 80% in this case. For other workloads, BATMAN has negligible changes of the 3D-DRAM access rate (within 1%).

5.2.4 Performance Improvement from BATMAN

Figure 5.6 shows the speedup of BATMAN with respect to the baseline in the cache mode. BATMAN improves the overall system bandwidth in two ways: First, the bandwidth of the Comm-DRAM becomes usable. Second, misses to the disabled sets that would have been misses in the baseline too would now avoid the 3D-DRAM bandwidth of miss-related

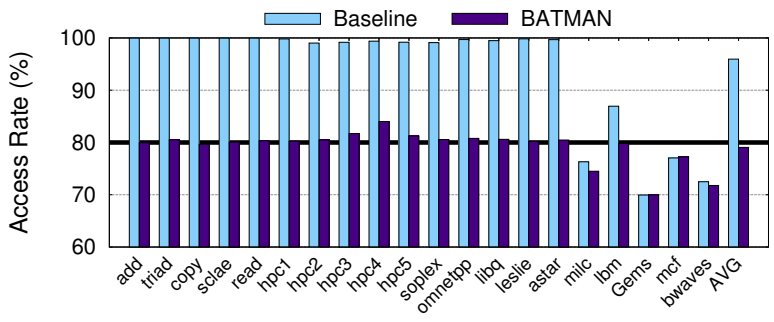

Figure 5.5: The 3D-DRAM access rate of the baseline and BATMAN

and writeback-related operations. As BATMAN improves overall system bandwidth, BATMAN provides an average speedup of 11%. More specifically, for applications whose working set fits in the cache, BATMAN consistently improves performance by as much as 23%. In addition, for applications whose working set is larger than the 3D-DRAM, BATMAN accurately captures and effectively reacts to the phases. For example, for *lbm*, adjusting the DSIndex captures different phases, shown in Figure 5.4(c), which results in 24% performance improvement.

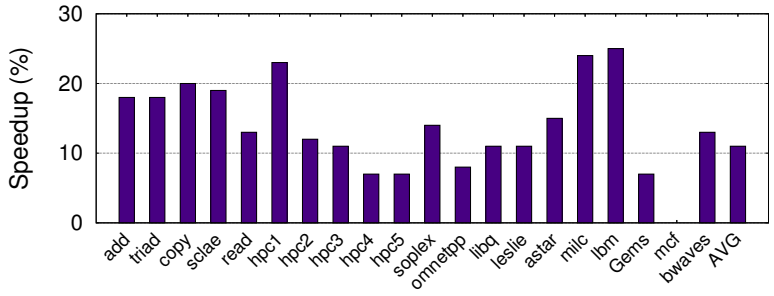

Figure 5.6: Speedup with BATMAN in the cache mode

5.3 BATMAN in the Flat Mode

BATMAN works for systems not only in the cache mode but also in the flat mode. Recall that in the flat mode, the system uses the 3D-DRAM as part of the memory space and relies on the operating system to performs dynamic page migration for data locality [17, 73]. Page migration relieves the system from being sensitive to the initial placement of

memory pages. For example, although a frequently access page may be initially placed in the Comm-DRAM, an access to the page transfers the accessed page from the Comm-DRAM to the 3D-DRAM, which allows subsequent memory accesses to the page to be serviced by the 3D-DRAM. When the size of the application working set is smaller than the capacity of the 3D-DRAM, a page migration scheme will eventually move the entire working set to the 3D-DRAM, thus underusing the Comm-DRAM bandwidth. Even when the working set does not fit in the 3D-DRAM, the page migration scheme would move frequently accessed data to the 3D-DRAM so that almost all memory requests are serviced by the 3D-DRAM. Ideally, the system migrates pages for data locality and yet ensure that the bandwidth utilization of both DRAM is balanced at the target access rate (TAR).

5.3.1 Idea: Regulate Direction of Page Migration

The idea of BATMAN that explicitly controls data movement to meet the TAR is applied to systems in the flat mode. In flat mode, page migration, in either direction is in the control of the OS. In the baseline system, the OS aggressively migrates a page from the Comm-DRAM to the 3D-DRAM on an access to the Comm-DRAM (also moves a page from the 3D-DRAM to the Comm-DRAM if 3D-DRAM is full). To control the data movement, BATMAN adds the constraint of the 3D-DRAM access rate when the OS is migrating the page because the 3D-DRAM access rate depends on the direction of the page migration: Migrating pages from the Comm-DRAM to the 3D-DRAM, referred to as *page upgrade*, increases the 3D-DRAM access rate; similarly, downgrading pages from the 3D-DRAM to the Comm-DRAM reduces the 3D-DRAM access rate. Therefore, BATMAN monitors the 3D-DRAM access rate at runtime and provides the information to the OS that decides the direction of page migration. Note that although the baseline assumes an aggressive page migration policy, BATMAN works with other policies, too. For example, for a frequency-based page migration strategy [17], BATMAN allows up to 80% accesses to be serviced by the 3D-DRAM.

5.3.2 BATMAN Design for Flat-Mode Systems

The key idea of BATMAN for flat-mode systems is to regulate the direction of page migration based on the 3D-DRAM access rate. If the 3D-DRAM access rate is lower than the TAR, BATMAN upgrades the accessed pages from the Comm-DRAM to the 3D-DRAM; otherwise, BATMAN downgrades pages from the 3D-DRAM to the Comm-DRAM. Figure 5.7 shows an overview of BATMAN for flat-mode systems. The default system has a 4x-bandwidth 3D-DRAM and a 1x-bandwidth Comm-DRAM, so the TAR is 80%. The system monitors the access rate of the 3D-DRAM and to decide to upgrade or downgrade pages, as described below:

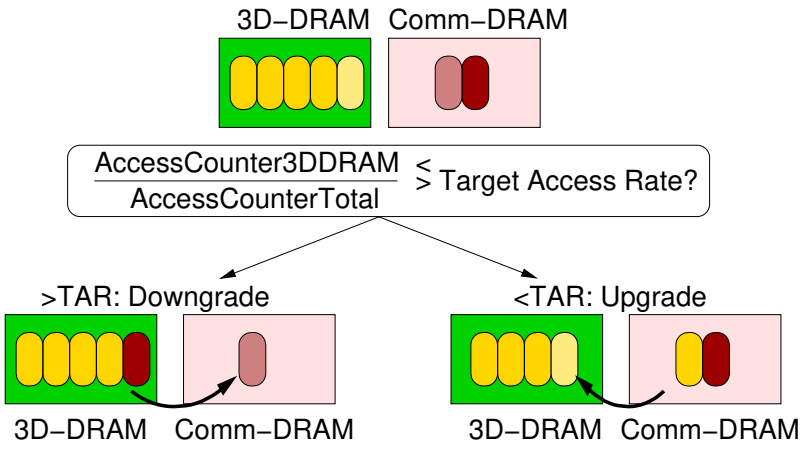

Figure 5.7: Overview of BATMAN in the flat mode: monitor the access rate of the 3D-DRAM and change the direction of page migration.

Structures

BATMAN dynamically monitors the fraction of total memory accesses that are serviced by the 3D-DRAM by using two counters: *AccesssCounter3DDRAM* and *AccessCounterTotal*, which count the number of accesses to the 3D-DRAM and to the system (both the 3D-DRAM and the Comm-DRAM), respectively. The ratio of the AccessCounter3DDRAM counter to the AccessCounterTotal counter shows the fraction of memory accesses serviced by the 3D-DRAM. Note that both these counters account for all memory activity and in-

crements on demand, prefetch, or writeback memory requests. BATMAN uses two 16-bit hardware registers for the counters. When the AccessCounterTotal counter overflows, both counters halve (right shift by one) their value. Therefore, BATMAN requires a storage overhead of only four bytes (two 16-bit counters) to track the accesses to the 3D-DRAM and the system.

Operation

On each access, the ratio of AccesCounter3DDRAM to AccessCounterTotal is computer. If this ratio is less than the TAR, BATMAN increases the access rate of the 3D-DRAM: If the memory request goes to Comm-DRAM, BATMAN upgrades the requested page from the Comm-DRAM to the 3D-DRAM. Similarly, if the ratio is greater than the TAR, BATMAN reduces the access rate of the 3D-DRAM: If the memory request accesses a page in the 3D-DRAM, BATMAN downgrades the requested page from the 3D-DRAM to the Comm-DRAM. BATMAN leverages the existing OS support for page migration between the 3D-DRAM and the Comm-DRAM. Regulating such downgrade and upgrade operations ensures that the 3D-DRAM access rate becomes close to the TAR. Note that BATMAN still utilizes all the memory capacity by downgrading a page to the Comm-DRAM, instead of evicting it to the storage.

Hysteresis on Threshold

Using a single threshold of the TAR leads to frequent switching between upgrade and downgrade modes. When the 3D-DRAM access rate is close to the TAR, the system can continuously switch between upgrade and downgrade operation. To avoid this oscillatory behavior, BATMAN provisions a guard band of 2% in either direction in the decision of upgrade and downgrade. Therefore, page upgrades occur only when the measured access rate of the 3D-DRAM is less than (TAR-2%) and downgrades occur only when the measured access rate of the 3D-DRAM exceeds (TAR+2%). In the intermediate zone, between

(TAR-2%) and (TAR+2%), BATMAN does not perform either upgrades or downgrades.

5.3.3 Effectiveness of BATMAN at Reaching TAR

Figure 5.8 shows the 3D-DRAM access rate for the baseline system with page migration and BATMAN. In the baseline, all workloads have a 3D-DRAM access rate close to 100%, even for the SPEC_BIG benchmarks suites whose working set is larger than the 3D-DRAM capacity because these workloads have high spatial locality within a page. For example, after an accesses to the Comm-DRAM upgrades a page to the 3D-DRAM, if the next 15 memory references go to other lines in the page, the 3D-DRAM access rate is as high as 15/16, or close to 94%. For other applications, as their working set fits in 3D-DRAM, all the pages are transferred to the 3D-DRAM and provides a 100% 3D-DRAM access rate. BATMAN redistributes some of the memory traffic to the Comm-DRAM, and balance the system based on the bandwidth ratio of the 3D-DRAM and the Comm-DRAM. For all workloads, BATMAN effectively obtains a 3D-DRAM access rate close to the target value of 80%. Thus, our proposed mechanism of BATMAN is effective at maintaining the 3D-DRAM access rate at the TAR, with a simple monitoring logic at runtime.

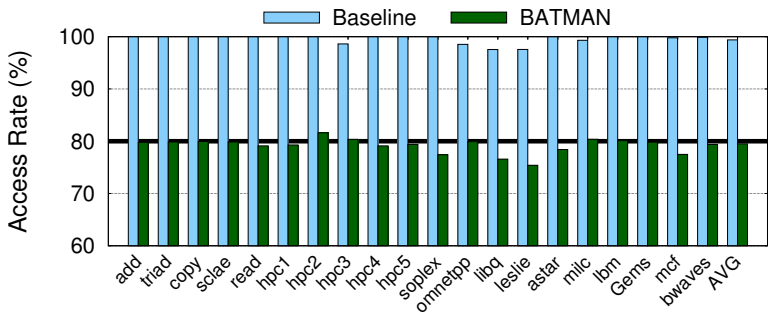

Figure 5.8: Access rate of the 3D-DRAM. BATMAN enforces the 3D-DRAM access rate close to the TAR (80%) for all workloads.

5.3.4 Performance Improvement from BATMAN

Figure 5.9 shows the speedup from BATMAN compared to the baseline system that always performs page migration between the 3D-DRAM and the Comm-DRAM on an ac-

cess to the Comm-DRAM. BATMAN improves performance of all workloads by an average of 10%. The performance improvement comes from two factors: First, for workloads whose working set fits in the 3D-DRAM, BATMAN effectively uses both 3D-DRAM and Comm-DRAM bandwidth and thus has higher throughput for those workloads. All HPC, STREAM and SPEC_SML (*soplex-astar*) workloads belong to this group. Second, for workloads form the SPEC_BIG category, whose working set is larger than the 3D-DRAM capacity, (e.g., bwaves-milc), BATMAN reduces the number of page migration between the 3D-DRAM and the Comm-DRAM because BATMAN allows only a certain number of page upgrades. As page migration (4KB transfer) consumes significant memory bandwidth on both the 3D-DRAM and Comm-DRAM, BATMAN reduces the bandwidth usage for this group of workloads. Thus, BATMAN improves performance regardless the working set of the application.

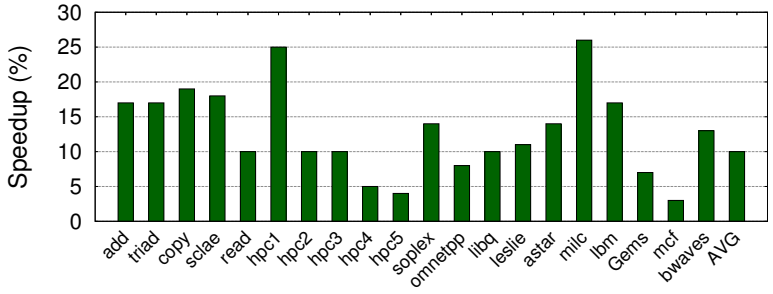

Figure 5.9: Speedup of BATMAN for systems in the flat mode

5.4 Methodology

5.4.1 System Configuration

The experiments models a 16-core system similar to one Intel's Knights Landing sub-NUMA cluster (SNC) node [31] by a detailed event-driven x86 simulator. Table 5.1 shows the core parameters, the cache hierarchy organization, and latency numbers, all of which are similar to the configuration of recent Intel Xeon processors [74]. Each core, running

at 3.2GHz, is a four-wide issue out-of-order processor with a 128-entry ROB. The on-chip cache subsystem contains a three-level cache hierarchy with private L1 and L2 caches and an L3 cache shared by the cores. The shared L3 cache is organized as a 16-way set-associative cache. All cache hierarchies use a 64B cache line.

Table 5.1: Baseline System Configuration for System-bandwidth Utilization Study

Processors	
Number of Cores	16
Frequency	3.2GHz
Core width	4-wide out-of-order
Prefetcher	Stream prefetcher
Last Level Cache	
Shared L3 cache	16MB, 16-way, 27 cycles
3D-DRAM	
Capacity	4GB
Bus frequency	800MHz (DDR 1.6GHz)
Channels	8
Bus width	64 bits per channel
tCAS-tRCD-tRP-tRAS	36-36-36-144 CPU cycles
Commodity DRAM (Comm-DRAM)	
Capacity	32GB
Bus frequency	800MHz (DDR 1.6GHz)
Channels	2
Bus width	64 bits per channel
tCAS-tRCD-tRP-tRAS	36-36-36-144 CPU cycles

The memory system consists of a 4GB 3D-DRAM using HBM2 technology [9] and a 32GB commodity DRAM (Comm-DRAM) using DDR3 technology [4]. As recent specification reveals that 3D-DRAM uses the same DRAM technology as DDR-based DRAM and thus has identical no-load latency, the study assumes the same timing parameters in both DRAM technologies [9, 5]. However, the bandwidth of the 3D-DRAM is higher than that of the Comm-DRAM. In the baseline system, the 3D-DRAM (4x channel) has 4x as high bandwidth as the Comm-DRAM. Note that this configuration is similar to one sub-NUMA cluster (SNC) node in Intel KNL. A sensitivity study of the bandwidth ratio is in Section 5.5. The DRAM simulator is similar to USIMM [64] and contains read

and write queues for each memory channel. The DRAM controller prioritizes reads over writes, and writes are issued in batches. The default memory address mapping policy is the minimalist-open page policy [75], which exploits memory channel parallelism and also retains the benefits of DRAM page hits. The policy places a group of four consecutive cache lines in the same DRAM page and interleaves groups among memory channels.

Cache mode In the first use case of the 3D-DRAM, 3D-DRAM is configured as a hardware-managed cache, referred to as *DRAM cache*, which is a direct-mapped, 64B-cache-line-size, and tags-with-data cache. The configuration of the DRAM cache is similar to that in a recent academic study [30] and that of commercial products [31]. The DRAM cache is equipped with a cache hit-miss predictor. The 3D-DRAM access rate in the cache mode includes all DRAM cache operations, such as miss- and writeback-related operations [69]. To optimize the hit rate, the baseline system in the cache mode installs the missed cache lines in the DRAM cache for future memory accesses.

Flat mode The system exposes 3D-DRAM and Comm-DRAM as two nodes to the OS, which determines the placement of memory page in and migrates memory pages between DRAM. The evaluation models a virtual memory system that translate the virtual memory address to the physical address and uses a 4KB page size. In this mode, the OS performs aggressive page migration that moves a page from the Comm-DRAM to the 3D-DRAM on an access to a page in the Comm-DRAM (and a page from the 3D-DRAM to the Comm-DRAM if the 3D-DRAM is full) [17, 73, 76]. The page migration overheads include reading page(s) from and writing to the DRAM, TLB shootdown, and OS management of page table. The simulation includes all the DRAM operations and uses a constant time of 1000 cycles for the OS overhead.

5.4.2 Workloads

The workloads are selected by Pin and SimPoints [77, 66] that capture a representative region of one billion instructions from each workload of various benchmark suites, includ-

ing SPEC CPU2006 [65, 78], STREAM [70], and high performance computing (HPC) workloads. The HPC workloads represents a wide range of workloads, including weather research and forecast (hpc1), high-performance computing cluster (hpc2), computational fluid dynamics (hpc3), in-cylinder flow and combustion (hpc4), and multi-purpose explicit and implicit finite element analysis (hpc5). The results show 20 workloads, including ten memory-intensive SPEC workloads, five STREAM and five HPC workloads. Table 5.2 shows the characteristics of the 20 workloads used in the study. Note that the working set size is the aggregate size of all 16 cores, which is observed during the simulation, unlike studies that report the resident set size (rss) and the virtual size (vsz) for the entire execution of workloads [78, 79]. The L3 MPKI reflects the bandwidth consumption of evaluated workloads, which is consistent with that of prior work [70, 79].

Table 5.2: Workload Characteristics for System-bandwidth Utilization Study

Category	Name	L3 MPKI	Aggregate Footprint(GB)
STREAM	add	83	3.58
	triad	73	3.58
	copy	71	2.38
	scalar	64	2.38
	read	43	2.38
HPC	hpc1	82	2.09
	hpc2	48	1.27
	hpc3	46	0.72
	hpc4	43	2.06
	hpc5	30	1.88
SPEC_SML	soplex	64	0.84
	omnetpp	50	2.19
	libq	48	0.50
	leslie	43	1.22
	astar	25	0.58
SPEC_BIG	milc	65	6.70
	lbm	64	6.23
	Gems	53	11.4
	mcf	43	18.5
	bwaves	39	6.61

5.4.3 Figure of Merit

The experiments execute benchmarks in the rate mode, in which all cores run the same benchmark. The SPEC workloads are classified into into two categories: Applications whose aggregated working set (for 16 cores) is larger than 4GB are categorized as SPEC_BIG; otherwise, they are categorized as SPEC_SML. The figure of merit measures the total execution time. As the workloads run in rate mode, the difference in execution time of the individual benchmark within the workload is negligibly small. The execution time is normalized to the baseline system of the respective modes. Also, as the goal of BATMAN is to control the 3D-DRAM access rate, the access rate of the 3D-DRAM, defined in Equation 5.1, is reported.

5.5 Results and Analysis

5.5.1 Sensitivity Study for Bandwidth Ratios

The default system assumes that 3D-DRAM has 4X bandwidth as Comm-DRAM, which is similar to the recent industry product [31]. A recent report indicates that the next generation of 3D-DRAM provides two lines of products [80]: high-end HBM3, which offers as approximately 8X high bandwidth as DDR4, and low-cost HBM2, which provides as 2X high bandwidth as DDR4. To study the sensitivity of BATMAN, this section varies the bandwidth ratio from 2X to 8X. As only the 3D-DRAM bandwidth changes, the study varies the bandwidth ratio by fixing the number of DRAM channels in the Comm-DRAM and varying the number of DRAM channels in the 3D-DRAM. That is, for an 8X ratio, the number of channels in the 3D-DRAM is as eight times as that in the Comm-DRAM. Table 5.3 shows the comparison of configurations. When the bandwidth ratio is 2X, the bandwidth increase from BATMAN is as high as 50%, which strongly suggests that when the Comm-DRAM bandwidth accounts for a significant fraction of overall system bandwidth, the system should optimize for overall bandwidth, not the 3D-DRAM access rate.

Table 5.3: Sensitivity Study of Bandwidth Ratio

3D-DRAM BW	Comm-DRAM BW	Utilized System BW		BW Increase
		Baseline	BATMAN	
2X	1X	2X	3X	**+50%**
4X	1X	4X	5X	+25%
8X	1X	8X	9X	+12.5%

Shown in Figure 5.10, when the ratio is 2X, BATMAN improves performance by more than 30% because the aggregate bandwidth is as 1.5 times high bandwidth as that of the 3D-DRAM, providing a huge increase in available bandwidth. When the ratio is 8X, the room for improvement is small because using the Comm-DRAM bandwidth increases total bandwidth by 12.5% and provides 9% and 5% speedup for systems in the cache and flat mode, respectively.

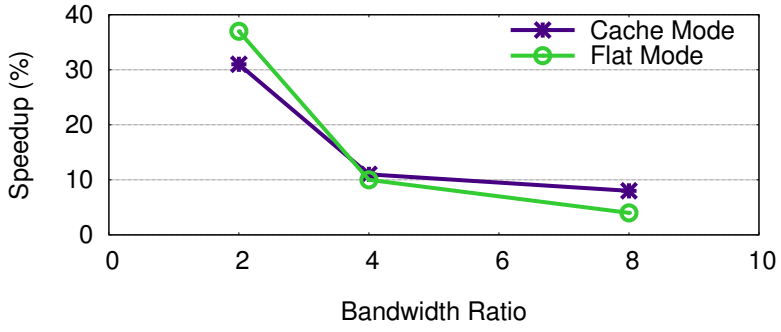

Figure 5.10: Sensitivity to the relative bandwidth of the Comm-DRAM: With fixed Comm-DRAM bandwidth, the 3D-DRAM bandwidth varies from 2X to 8X. Each configuration is normalized to the respective baseline. When the 3D-DRAM bandwidth is as 2X high as the Comm-DRAM bandwidth, BATMAN improves overall bandwidth by 50% and provides more than 30% speedup.

5.5.2 Power and Energy Analysis

As BATMAN utilizes the idle Comm-DRAM bandwidth, BATMAN precludes some power-saving techniques that are applied in the idle mode. To understand the implication of BATMAN in power consumption, this section a analyzes the power consumption and the energy-delay product (EDP) for both the cache and flat modes. The Comm-DRAM is

DDR3 technology [81], whose power is calculated based on the Micron DDR3 DRAM power calculator. The 3D-DRAM is based on Micron 3D-DRAM technology [8]. For the power measurement, this analysis conservatively assumes the memory system consumes 30% of the system power and the remaining system consumes 70% of the system power [82]. For systems that have high memory intensity, the memory system may consume more than 30% of the system power. Although this study conservatively assumes 30%, BATMAN could further improve system energy and energy-delay product if the memory system consumes more power.

Figure 5.11 shows the power consumption and the energy-delay product for the baseline and BATMAN in two modes. Note that each configuration is normalized to the respective baselines. With BATMAN, the overall power consumption of the system increases by 7% and 10% for systems in the cache and the flat mode, respectively. The power consumption contributed by BATMAN comes from two reasons. First, BATMAN reduces the execution time, which increases power because the all system activity must happen within a reduced time interval. Second, BATMAN exploits the Comm-DRAM bandwidth in the system and incurs the active power of the Comm-DRAM. However, lower execution time reduces energy consumption; BATMAN reduces the energy consumption by 5% and 1% for two modes, respectively. Also, BATMAN improves the EDP of the system by 13% and 11% for two modes, respectively.

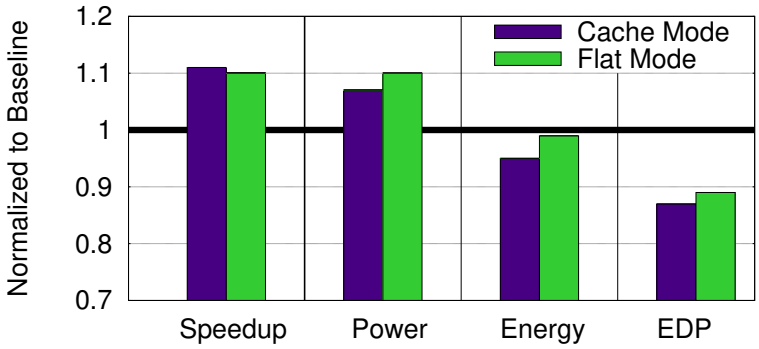

Figure 5.11: Speedup, power, energy, and EDP with BATMAN (numbers normalized to respective baseline)

5.6 Summary

Emerging 3D-DRAM technology, such as HBM and HMC, provides as 4x to 8x high bandwidth as the commodity DDR-based DRAM. The technology is used in tiered-memory systems, in which conventional management approaches focus on improving the number of memory requests serviced by the 3D-DRAM. However, such techniques under-utilize the commodity DRAM bandwidth, especially when the frequently-accessed working set fits into the 3D-DRAM. This dissertation shows that system bandwidth and performance are maximized when memory accesses are split between the 3D-DRAM and the commodity DRAM proportional to the bandwidth of each DRAM. The key insight is that the control of data movement regulates the 3D-DRAM access rate. To achieve a desired access distribution, this dissertation proposes a runtime mechanism, referred to as *bandwidth-aware tiered-memory management (BATMAN)*, which explicitly controls the data movement between the 3D-DRAM and commodity DRAM.

BATMAN demonstrates its effectiveness on both cache- and flat-mode systems. In the cache mode, BATMAN tracks the DRAM cache access rate at runtime and disables a fraction of the cache sets to obtain the target access rate. Adjusting the DSIndex at runtime regulates the DRAM cache access rate and also adapts to dynamic phases of workloads. In the flat mode, BATMAN monitors the 3D-DRAM access rate at runtime and dynamically controlling the direction of page migration, which is highly effective at reaching the target access rate. The implementation of BATMAN is simple and highly effective at maintaining the 3D-DRAM access rate at the TAR. BATMAN incurs a storage overhead of only eight bytes and requires negligible software support. The experiments on a 16-core system with a 4GB 3D-DRAM and a 32GB commodity DRAM show that BATMAN improves performance for systems in the cache and flat mode by an average of 11% and 10%, respectively; also, BATMAN improves the system energy-delay-product by 13% for systems in the cache mode and 11% for systems in the flat mode.

CHAPTER 6

CACHE-LIKE MEMORY ORGANIZATION: A FINE-GRAINED TRANSPARENT TWO-LEVEL MEMORY ARCHITECTURE

Although DRAM cache has the advantage that it can be deployed without relying on OS support, the capacity of 3D-DRAM is not visible to the operating system (OS). Therefore, DRAM cache works well when the capacity of 3D-DRAM, compared to the commodity DRAM, is sufficiently small. As the technology for manufacturing 3D-DRAM matures, its size ultimately accounts for a quarter or even half of the overall capacity in the memory system, which makes the DRAM cache less attractive. As a result of the loss of memory capacity, applications with large memory footprints can suffer higher page fault rate and thus slowdown because of frequent storage accesses. Figure 6.1 shows the performance improvement of DRAM cache, which is one quarter of total DRAM capacity. The workloads are classified into two categories: capacity-Limited (memory footprint more than off-chip memory) and latency-Limited (memory footprint fits in off-chip memory). Overall, DRAM cache provides 50% improvement; however, for capacity-limited workloads the improvement is marginal because the OS does not account the capacity of 3D-DRAM.

Instead of a DRAM cache, two-level memory (TLM) has the advantage of providing an effective capacity that is the sum of both 3D-DRAM and commodity DRAM. This allows the system to accommodate a larger number of pages, which reduces the number of page faults (which may incur latency as high as 10^5 to 10^6 cycles). Also, the advantage of TLM is that it avoids the tag store overhead of a cache, as it leverages the existing paging mechanism to decide the physical location of the page. A simple way to deploy TLM is to statically partition the address space into high-bandwidth 3D-DRAM region and low-bandwidth commodity DRAM region This design is referred to as *TLM-Static*, which is oblivious to the characteristics of different memories and randomly maps the pages across

the memory address space. Figure 6.1 shows the performance of TLM-Static. As the total memory capacity visible to the OS gets increases, the capacity-limited workloads see significant benefits (67% speedup on average). However, when workloads are not limited by memory capacity, the benefits are much reduced (18% versus 82% of DRAM cache) because DRAM cache retains the data lines with high locality in the 3D-DRAM, whereas TLM-Static does not optimize for data locality.

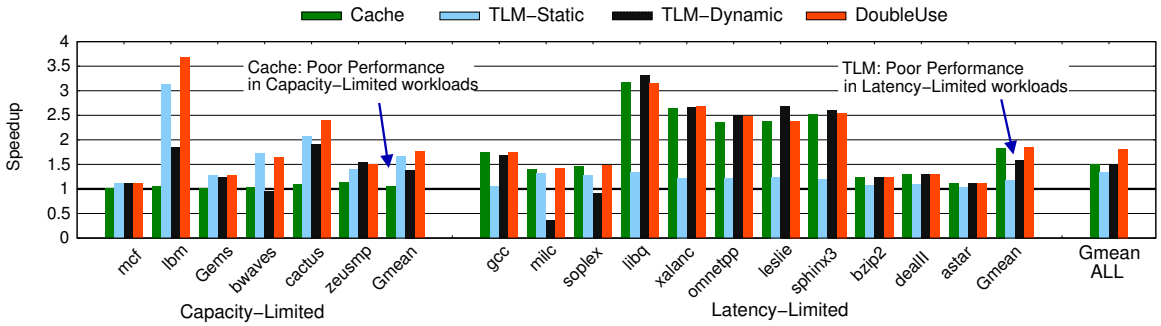

Figure 6.1: Performance evaluation of a system, where 3D-DRAM is one quarter of total DRAM capacity, implemented as a hardware-managed cache, or two-level memory (with and without page migration), or an idealistic "DoubleUse" system that uses 3D-DRAM as a hardware-managed cache and increases memory capacity by the size equivalent to the 3D-DRAM.

6.1 Motivation: The Problem of Coarse-grained Migration

The OS optimizes TLM for data locality by migrating pages with high locality from commodity DRAM to 3D-DRAM. Such an optimization is referred to as *TLM-Dynamic*, which retains recently accessed pages in 3D-DRAM. It does so by swapping a page, accessed in commodity DRAM, with a victim page in 3D-DRAM. Unfortunately, such data migration must occur at a page granularity (4KB in typical systems). The cost of migrating data at page granularity is very high, as it entails a swap operation of 4KB between both DRAM. Both DRAM modules must read and write the respective 4KB pages (a total memory activity of 16KB). As not all lines in a page get used, page granularity transfers are highly inefficient for memory bandwidth. Figure 6.1 shows the performance of TLM-Dynamic. for capacity-limited workloads, the overhead of data migration far outweighs the potential ben-

efits. However, for latency-limited workloads, TLM-Dynamic degrades the performance of some workloads. Overall, TLM-Dynamic has better performance than TLM-Static (50% versus 33%). Although data migration optimizes for locality, doing so at large granularity may limit the performance.

A desirable architecture should provide full memory capacity without the OS support and still optimize data locality at a fine granularity. To illustrate the performance of such a design, an "idealistic" configuration, referred to as *DoubleUse*, uses 3D-DRAM as a hardware cache but also increases the capacity of commodity by the size of 3D-DRAM. In essence, this is a theoretical configuration to show the potential improvement possible with having increased memory capacity and performing fine-grained data migration. Figure 6.1 shows the performance of the DoubleUse system. For latency-limited workloads, DoubleUse performs similar to hardware cache, as these workloads do not need higher memory capacity. However, for capacity-limited workloads DoubleUse performs significantly better than hardware cache, and marginally better than TLM. Overall, the DoubleUse system has a performance of 82%, whereas optimizing the system only for capacity (TLM-Static) provides 33%, optimizing for both capacity and locality at page granularity (TLM-Dynamic) provides 50%, and hardware cache provides 50%. The goal is to develop an organization that has cache-like properties of managing data at fine granularity, while still providing full memory capacity without relying on the OS support.

6.2 CAMEO: Architecture and Design

While a two-level memory optimizes for high capacity, and a cache optimizes for fine-grained data movement via hardware management. To achieve both objectives simultaneously, this dissertation proposes *cache-like memory organization (CAMEO)*. Figure 6.2 provides an overview of CAMEO. In the example, the 3D-DRAM has capacity of N lines, and the commodity DRAM memory has capacity of 3N lines. Combining both DRAM would provide a visible address space of 4N lines. For simplicity, let's assume that the

memory space starts from 3D-DRAM and grows to the region of commodity DRAM. To leverage locality like hardware cache, CAMEO keeps the recently accessed data line in stacked memory by swapping data lines between the two memory regions. The group of lines that can be mapped to a given location in 3D-DRAM is referred to as a *congruence group*. For example, lines A, B, C, and D form a congruence group. The restriction is that lines can only be swapped with another line from the same congruence group. This is similar to the group of lines contending for the same set in a hardware cache. The number of congruence groups is equal to the number of lines in 3D-DRAM. If there are N lines in 3D-DRAM, the bottom $log_2(N)$ bits of the requested line address identifies the congruence group of the line.

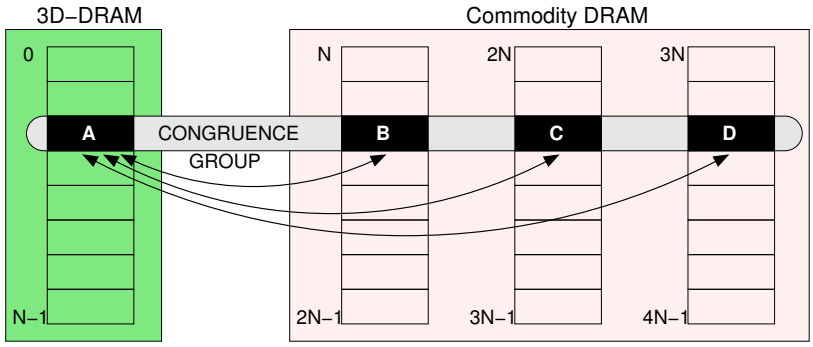

Figure 6.2: Overview of CAMEO: Lines A, B, C and D form a *congruence group*. CAMEO performs swapping only within the congruence group.

When a line in the commodity DRAM is accessed, say line B, CAMEO would evict line A from 3D-DRAM and store line B in the location of line A. CAMEO would then store line A in the commodity DRAM where line B was initially stored. Swapping maximizes effective capacity by ensuring that there is only one copy of the line in main memory. **Line Swapping:** The swapping operation is performed in hardware using existing writeback and fill queues. As CAMEO operates on line granularity, a swapping operation is done as a writeback from 3D-DRAM to commodity DRAM, and a demand read from commodity DRAM to 3D-DRAM.

6.2.1 Line Location Table

CAMEO performs the swapping operation in hardware in a manner that is transparent to the operating system. Such swapping mandates that a given line relocates to another position within the congruence group. To correctly identify the position of a requested line, CAMEO must track the physical position of all data lines. The hardware structure that keeps track of this information is referred to as the *line location table (LLT)*. For each congruence group, the LLT keeps a record of the physical location of all the lines. The address requested by the LLC of the processor is termed the *requested address* and the real address where such line is located the *physical address*. As LLT is kept at the granularity of a congruence group, each LLT entry provides a mapping of all the lines in the congruence group. For example, for our configuration with 4GB 3D-DRAM and 12GB commodity DRAM, four lines form a congruence group, and each entry in the LLT will be a four-entry tuple with two bits of location for each of the four lines in the congruence group.

Figure 6.3 illustrates the operation of LLT with an example. Lines A, B, C, and D belong to the same congruence group. Initially, the LLT entry contains an identity mapping where the physical addresses of the lines are identical to the real location. When a request to line B is made, CAMEO swaps Line A and B, and records the new mapping in the LLT. When a subsequent request is made to say line D, CAMEO would swap line B (which is in 3D-DRAM) with line D, and update the LLT entry accordingly. Thus, a line can move to any location within the congruence group (for example, line B got moved to the commodity DRAM at the location of line D). CAMEO uses the LLT much like the "tag-store" in a traditional cache to identify which line is resident in the 3D-DRAM. However, unlike hardware cache, CAMEO also uses the LLT information to identify the real location of the line in the commodity DRAM in case the line is not found in the 3D-DRAM.

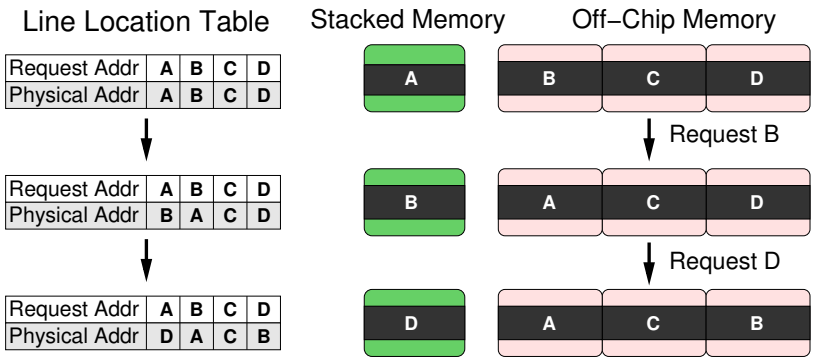

Figure 6.3: Operation of the Line Location Table (LLT), which keeps location information for each Congruence Group. Lines A, B, C, and D form a Congruence group, and operation of LLT is shown after two memory requests are performed.

6.2.2 Design Challenges for the Line Location Table

To access a line in CAMEO, the request must first access the line location table (LLT) to determine the physical location of the line. Only then can the memory controller decide whether the line should be obtained from the 3D-DRAM or commodity DRAM. To keep CAMEO practical, it is important that the storage and latency overheads of the LLT are kept to a minimum. However, this is a challenging task. For a system with 4GB 3D-DRAM and 12GB commodity DRAM, each congruence group will have four lines. Thus, each LLT entry will be one byte (4 entries of 2 bits each). For a 16GB system, there would be 64 Million congruence groups (16GB divided by 256B, the size of the congruence group). Thus, the total size of the LLT for such a system is 64 MB, which is prohibitively large for an on-chip store. The next discusses the design trade-offs and challenges in architecting such a large LLT.

SRAM-Based LLT (Impractical)

The size of LLT (64MB) is greater than the size of the last-level cache (LLC) in current microprocessors. Therefore, designing a LLT made of SRAM would incur unacceptably high overhead (in essence, sacrificing the L3 cache for storing LLT). Furthermore, accessing the LLT would still incur a latency overhead of as high as the L3 cache (24 cycles).

Figure 6.4(a) shows the design of SRAM-based LLT. The requested line address probes the LLT to identify the real location of the line and accesses the physical location of this line address for data. As this design incurs impractically high storage overhead, this design is only of theoretical importance and not considered any further.

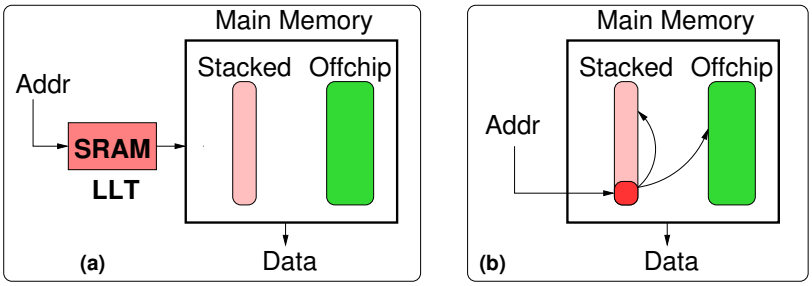

Figure 6.4: Options for LLT. (a) SRAM-based LLT incurs impractical storage overhead (b) LLT can be embedded in 3D-DRAM but incurs indirection latency (first access for LLT, second for data

Embed LLT in 3D-DRAM (Practical but Slow)

A more practical approach to design the LLT is to avoid the SRAM overheads by storing the LLT in the 3D-DRAM. Figure 6.4(b) shows such a design that embeds the LLT in 3D-DRAM. A portion of 3D-DRAM is reserved to serve as the LLT. An incoming line address first indexes the LLT to obtain the LLT entry. Based on the real location of the line, then the second access is performed for obtaining data, from either the 3D-DRAM or off-chip DRAM. This approach, referred to as *embedded-LLT*, sacrifices some capacity of 3D-DRAM for storing LLT and makes this capacity invisible to the memory address space. Fortunately, the size of the LLT (64MB) is much smaller than the size of the 3D-DRAM (4GB). The first 64MB of 4096 MB 3D-DRAM is reserved for the LLT, and the remaining 4032 MB is available to serve as main memory. Thus, with embedded-LLT, 98.5% of the stacked memory is still available to serve as main memory. Unfortunately, embedded-LLT introduces the indirection latency of looking up the LLT before accessing data. This latency overhead increases the effective latency of memory accesses and degrades performance.

6.2.3 Practical LLT by Co-location with Data Line

In the common case, the memory requests will be serviced by 3D-DRAM. Therefore, removing the serialization latency of LLT lookup for the lines that are resident in the stacked DRAM can significantly improve the latency CAMEO uses a design, referred to as the *co-Located LLT*, which co-locates the LLT entry with the data line. Each data line is appended with a LLT entry to form an entity called *location entry and data (LEAD)*, which is the basic unit of one 3D-DRAM access. If the LLT entry in the LEAD identifies that the requested line is present in the 3D-DRAM, the data in the LEAD directly returns to the processors without any extra access to the 3D-DRAM. If the LEAD identifies that the line is in the off-chip memory, a second access to the desired location in off-chip memory is performed. Thus, co-Located LLT can avoid the LLT lookup serialization for lines that are resident in the 3D-DRAM.

The row buffer of the 3D-DRAM used in the study is 2KB. The space for the co-Located LLT at the granularity of a LEAD is one data line, which supports the location table entry for the other 31 lines. Thus, each LEAD can have up to two bytes of location table entry (one byte used and one byte reserved for future use). The size of one LEAD is thus, 2+64=66 Bytes. The size of the data bus for the stacked DRAM used in the studies is 16 bytes, so a burst length of five transfers 80 bytes. LEAD uses 66 bytes out of 80 bytes and ignores the extra 14 bytes. Figure 6.5 shows the design of the co-Located LLT for a given row buffer in 3D-DRAM. The 2KB row buffer can accommodate 31 units of LEAD, resulting in a useful capacity of 31/32 (97%). For simplicity, the first 32MB in memory space in not visible to the operating system. Before accessing the 3D-DRAM, the physical address is modified appropriately. For a given LineAddr X in stacked memory, the revised location of 3D-DRAM is obtained using [(X + X/31)-LinesIn32MB]. Note, the division operation can simply be performed with a few adders using residue arithmetic (31=32-1). This operation can be performed in parallel with the L3 access to hide the latency.

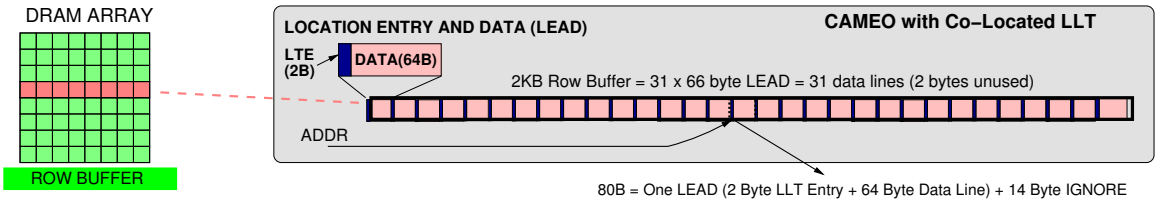

Figure 6.5: Organization of co-Located LLT. The LLT entry (LTE) is co-located with the data line to form a location entry and data (LEAD) of 66 bytes. The 2KB row buffer stores 31 LEAD. Each access to 3D-DRAM provides one LEAD.

6.2.4 Latency Comparisons of LLT Designs

An ideal design of LLT, termed *ideal-LLT*, incurs zero overheads for LLT storage and latency. As soon as the memory controller receives the requested line address, it knows the real location and accesses the data from that location. Figure 6.6 compares the latency of embedded-LLT and co-Located LLT to ideal-LLT. An access to 3D-DRAM incurs 1 unit of latency and an access to off-chip DRAM incurs two units. The analysis shows a single memory request serviced in isolation. For the baseline (no 3D-DRAM), the request, serviced by off-chip DRAM, incurs a latency of 2 units. With an ideal-LLT, if the line is in 3D-DRAM (case denoted as H), it is serviced with a latency of one unit; if the line is in off-chip DRAM (case denoted as M), it is serviced with a latency of two units. For the embedded-LLT, the LLT lookup takes one unit of time. The data line in 3D-DRAM is serviced in two units (case H), and the data line in off-chip DRAM takes 3 units (case M). Thus, embedded-LLT has no latency advantages for accessing data from 3D-DRAM (albeit there may still be bandwidth benefits), and a slowdown for off-chip accesses. For the co-Located LLT, the data line in 3D-DRAM is serviced in 1 unit (LLT access and data access happen in one transfer); if the data is in off-chip DRAM, the lookup latency of LLT becomes serialized, and the total latency is 3 units. Thus, co-Located LLT has lower latency for data lines in the 3D-DRAM; however, it has higher latency for off-chip accesses.

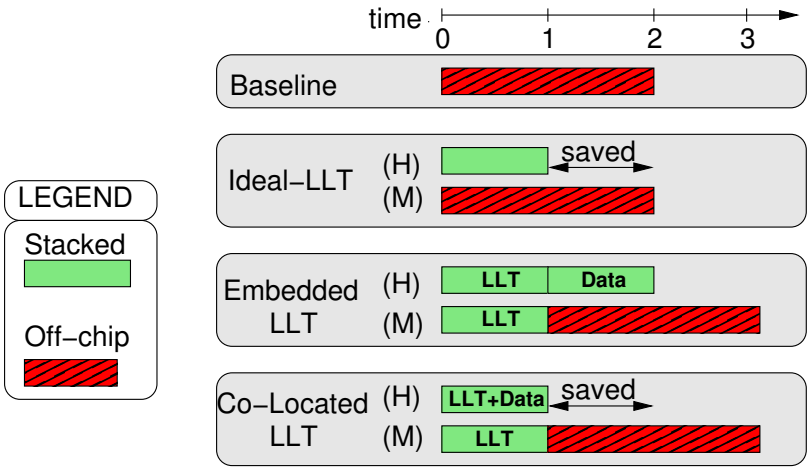

Figure 6.6: Access Latency Comparison for different LLT designs, for a system with 3D-DRAM latency of 1 unit and off-chip DRAM latency of 2 units.

6.2.5 Performance Comparisons of LLT Designs

Figure 6.7 compares the speedup of CAMEO with ideal-LLT, embedded-LLT, and co-Located LLT. As CAMEO provides a high memory capacity, there are benefits for capacity-intensive workloads for almost all CAMEO configurations. Embedded-LLT, with high latency overheads, results in performance slowdown for latency-sensitive workloads. Co-Located LLT has lower latency when data lines are resident in stacked DRAM, and thus improves the performance by an average of 74%. However, a significant performance gap between co-Located LLT and ideal-LLT (on average, 74% versus 80%) still remains because because of the serialization of LLT looko-ups for lines in off-chip DRAM. The next section describes solutions to avoid the serialization of LLT look-ups.

6.3 Memory Location Prediction

The co-Located LLT avoids the latency of LLT look-ups for lines resident in the 3D-DRAM by fetching the LLT entry and data together. However, it still suffers from the latency of LLT look-up for lines that are resident in the off-chip DRAM. Such serialization overhead can be avoided by a prediction mechanism that speculatively issues a off-chip request. The

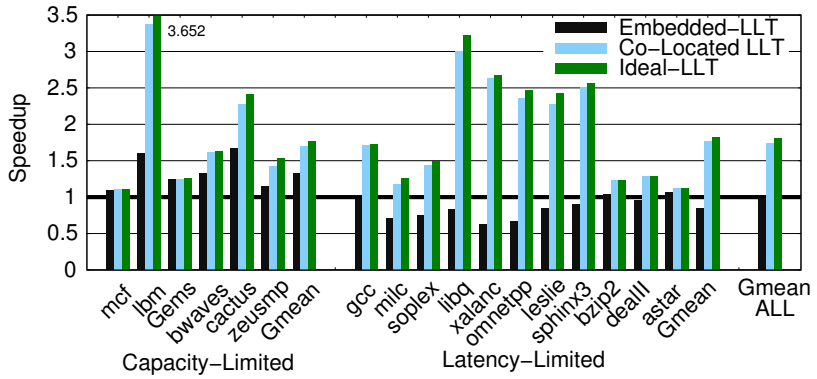

Figure 6.7: Speedup of different LLT designs. Embedded-LLT has high latency overheads, hence the slowdowns. Co-Located LLT has low latency for data lines in 3D-DRAM; however, because of higher off-chip latency the performance is lower than ideal-LLT.

section first describes the framework for how such a predictor can be integrated in CAMEO and the design of the predictor, followed by performance evaluation.

6.3.1 Avoiding LLT Latency with Location Prediction

Figure 6.8(a) shows the memory access with CAMEO. The serialized off-chip DRAM access happens only after accessing the 3D-memory. Such a model of memory access is referred to as *Serial Access Memory (SAM)*. Alternatively, a speculation predicts the physical location of the line using a *line location predictor (LLP)*. The organization of CAMEO with LLP is shown in Figure 6.8(b). If the LLP predicts that the location of the line is in off-chip DRAM, CAMEO accesses both the 3D-DRAM and off-chip DRAM in parallel. Only the predicted location in off-chip DRAM is accessed. If the line is found in 3D-DRAM, then the data from off-chip DRAM are ignored. However, if the line is not found in 3D-DRAM, the location provided by the LLP is verified with the LLT entry obtained from the 3D-DRAM. If the prediction is correct, the line from the off-chip location is used. Given that the off-chip access was made in parallel with 3D-DRAM access, this scheme avoids the latency of LLT look-ups.

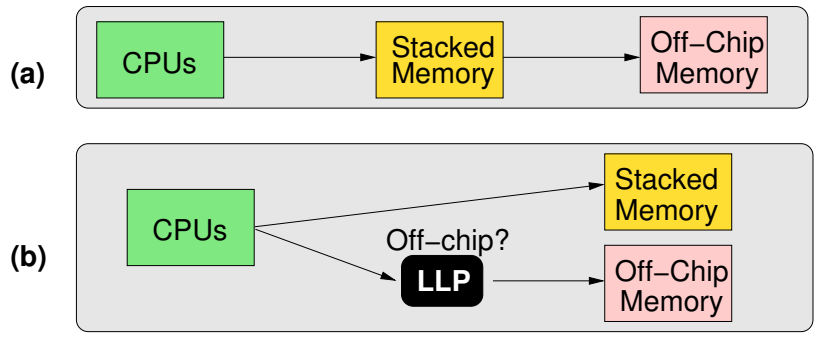

Figure 6.8: Avoiding LLT latency with prediction. (a) With SAM, off-chip access happens only after 3D-DRAM access (b) If access is predicted to be off-chip, the predicted location is accessed in parallel.

6.3.2 Line Location Predictor

The effectiveness that avoids the serialization of LLT look-up for off-chip accesses depends on the accuracy of the LLP. The key challenge for designing an effective LLP is that the LLP must decide upon the correct location from multiple candidate locations. This is unlike previous schemes on cache hit prediction [30, 67] that makes a binary decision between cache and memory. In our configuration, the line could be in any of the four locations, say 00, 01, 10, or 11, as location 00 being in 3D-DRAM, and other three locations being in off-chip DRAM. Thus, the LLP must make a prediction out of four choices, as shown in Figure 6.9(a). As memory references are known for a good correlation with past behavior [83], the predictor exploits the history in memory reference stream. In particular, the LLP employs *last time prediction*: the LLP will predict a location same as it provided the last time. A simple implementation of such a history-based last time predictor is to keep a two-bit register called *line location register (LLR)*, which tracks the physical address of the recent L3 miss. On the next L3 miss, if the location in LLR is in 3D-DRAM (location 00), the predictor uses serial access. Otherwise, the predictor uses the speculated location provided by LLR and accesses the off-chip location within the congruence group.

Further enhancement of the LLP is based on the observation that the memory reference stream tends to be heavily correlated with the instruction address that issues the memory

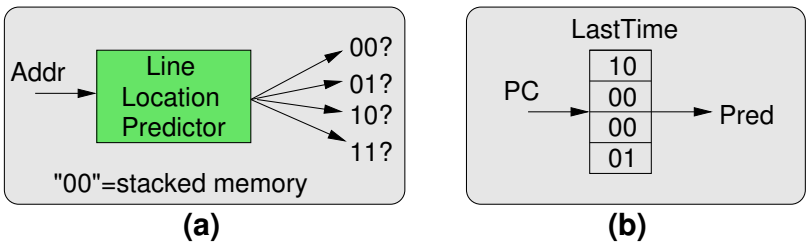

Figure 6.9: Line Location Predictor (a) LLP must make a 4-ary choice (b) A PC-based LLT implementation that predicts the location based on last-time.

access [55, 57]. Instead of a single LLR, the LLP employs a table of LLRs, indexed by the instruction address of the L3-miss-causing instruction. A sensitivity study shows that using a 256-entry (8-bit index) table is quite effective at bridging the performance gap between serial access and perfect prediction. As each LLR is 2-bits, a table of LLR with 256 entries, requires 64 bytes. The tables of LLRs are on a per-core basis, so eight such prediction tables are employed, incurring a total storage overhead of 512 bytes. Thus, the line location predictor requires negligible overheads.

6.3.3 Prediction Accuracy Analysis

Assessing the accuracy of LLP requires the understanding of five possible cases that can occur: 1) The physical location is in 3D-DRAM, and the predictor predicts it as such. 2) The physical location is in 3D-DRAM, but the predicted location is in off-chip DRAM. 3) The physical location is in off-chip DRAM, but the predictor gives a location in 3D-DRAM. 4) The physical location is in off-chip DRAM, and the predicted location is correct and in off-chip DRAM. 5) The physical location is in off-chip DRAM, and the predicted location is not correct but still in off-chip DRAM. The LLP makes accurate prediction in case 1 and 4, while case 2, 3, and 5 are deemed as mis-prediction. Although case 2, 3, and 5 are mis-predicted, they have different consequence in terms of latency and bandwidth. Case 2 wastes off-chip DRAM bandwidth, case 3 increases the effective latency, and case 5 is a combination of bandwidth waste and latency increase. Table 6.1 shows the percentage of each scenario for no prediction (serial access), prediction using LLP, and perfect predictor.

SAM has 70.3% accuracy, which means higher latency for 29.7% accesses. LLP has an accuracy of 92%, meaning for the 92% requests LLP is accurate, and provides both low latency and avoids wasteful bandwidth from parallel access. For the rest of the chapter, CAMEO is implemented with LLP.

Table 6.1: Accuracy of Line Location Predictor

Serviced by	Prediction	SAM	LLP	Perfect
3D-DRAM	3D-DRAM	70.3	**68.4**	70.3
	Off-chip	0	1.8	0
Off-chip	3D-DRAM	29.7	1.7	0
	Off-chip (OK)	0	**23.3**	29.7
	Off-chip (Wrong)	0	4.8	0
Overall Accuracy		70.3	91.7	100

6.3.4 Performance Results of LLP

Figure 6.10 shows the speedup for CAMEO using co-Located LLT, with and without the LLP predictor. It also compares the performance with a prefect predictor that has 100% accuracy. On average, the performance improvement with SAM is 74%, and with perfect predictor is 80%. The line location predictor provides an average performance of 78%, within 2% of a perfect predictor. Thus, even though the proposed implementation is simple and low overhead, it is still highly effective at obtaining most of the potential performance from location prediction.

6.4 Methodology

6.4.1 System Configuration

The experiments are conducted on a Pin-based [77] x86 simulator with a detailed memory system model. Table 6.2 shows the configuration used in the study. The parameters for the L3 cache, and DRAM (both 3D-DRAM and off-chip DRAM) are similar to the recent studies on 3D-DRAM [21, 30]. DRAM cache is based on alloy cache [30]. For the heterogeneous memory system 3D-DRAM accounts for 25% of the total DRAM capacity:

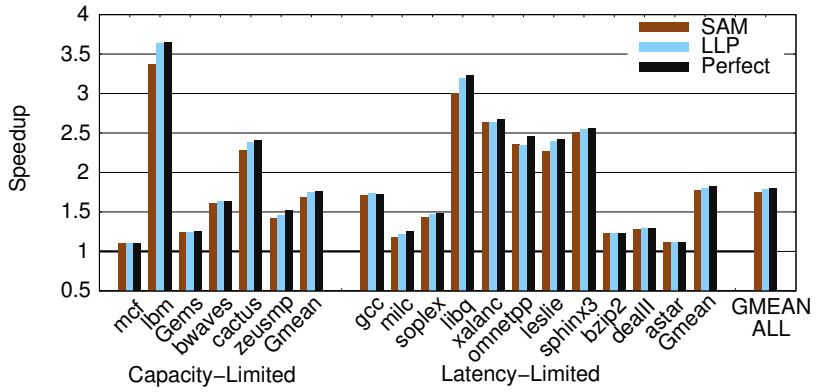

Figure 6.10: Speedup for no prediction, location prediction, and perfect prediction. On average, no prediction provides 68%, LLP provides 89%, and perfect prediction provides 94%.

12GB off-chip DRAM provisioned with 4GB of 3D-DRAM. A virtual to physical memory address translation handles the page faults, and the victim page is found using a clock algorithm, if an invalid page is not available (after five random tries). Page faults are serviced by a solid-state disk with a latency of 32 microsecond (10^5 cycles) [84].

6.4.2 Workloads

The workloads are chosen from a representative slice of 20-billion instructions from SPEC CPU 2006 suite [65]. The evaluation is performed by executing benchmarks in rate mode, where all cores execute the same benchmark. Given that the study is about the memory system, workloads that spend a negligible amount of time in memory are not meaningful for our studies. To capture the memory system activity for different applications, benchmarks are classified based on memory working set and miss per thousand instructions (MPKI) in L3 cache. As the baseline system has 12GB memory, benchmarks that have a working-set size larger than 12GB are referred to as *capacity-limited* workloads. The remaining benchmarks (working set less than 12GB) are sorted based on MPKI and benchmarks with MPKI greater than 1 are grouped as *latency-limited* workloads. Although the results of the remaining SPEC benchmarks (working set less than 12GB, and MPKI less than 1) are not shown, cache, TLM, and CAMEO have similar performance for these workloads.

Table 6.2: Baseline System Configuration for the CAMEO study

Processors	
Number of Cores	32
Frequency	3.2GHz
Core Width	2 wide out-of-order
Last Level Cache	
Shared L3 Cache	32MB, 16-way, 24 cycles
Stacked DRAM	
Bus Frequency	1.6GHz (DDR 3.2GHz)
Channels	16
Banks	16 Banks per rank
Bus Width	128 bits per channel
tCAS-tRCD-tRP-tRAS	9-9-9-36 bus cycles
Off-Chip DRAM	
Bus Frequency	800MHz (DDR 1.6GHz)
Channels	8
Banks	8 Banks per rank
Bus Width	64 bits per channel
tCAS-tRCD-tRP-tRAS	9-9-9-36 bus cycles
SSD Storage	
Page Fault Latency	32 micro seconds (100K cycles)

Table 6.3 shows the L3 MPKI and memory footprint for the workloads used in the study. The virtual-to-physical mapping ensures that multiple benchmarks do not map to the same physical address.

6.4.3 Figure of Merit

The execution time is measured when all benchmarks in the workload finish execution (as benchmarks are run in rate mode there is negligible variation in completion time of different benchmarks within a workload). The speedup of a given configuration is reported as the execution time of that configuration normalized to the baseline with no 3D-DRAM.

Table 6.3: Workload characteristics (32-copies in rate mode) for the CAMEO study

Limited By	Name	L3 MPKI	Memory Footprint
Capacity	mcf	39.1	52.4GB
	lbm	28.9	12.8GB
	GemsFDTD	19.1	25.2GB
	bwaves	6.3	27.2GB
	cactusADM	4.9	12.8GB
	zeusmp	5.0	14.1GB
Latency	gcc	63.1	2.8GB
	milc	31.9	11.2GB
	soplex	28.9	7.6GB
	libquantum	25.4	1.0GB
	xalancbmk	23.7	4.4GB
	omnetpp	20.5	4.8GB
	leslie3d	15.8	2.4GB
	sphinx3	13.5	0.60GB
	bzip2	3.48	1.1GB
	dealII	2.33	0.88GB
	astar	1.81	0.12GB

6.5 Results

6.5.1 Performance

Figure 6.11 shows the speedup from using 4GB 3D-DRAM as either a hardware-managed cache, or two-level memory (static and dynamic), or CAMEO (with co-Located LLT + LLP). It also compares these designs with an idealistic configuration (*DoubleUse*) that uses the 4GB as a hardware-managed cache, but also increases the size of commodity DRAM by an additional 4GB. On average, cache provides an improvement of 50%, TLM-Static provides 33%, TLM-Dynamic provides 50%, CAMEO provides 78%, and DoubleUse provides 82%. In summary, CAMEO provides both memory capacity of TLM, and fine-grained management of cache and thus outperforms both designs. On average, the performance of CAMEO is very close to the performance of the idealistic DoubleUse configuration that not only uses the 4GB 3D-DRAM both as hardware-managed cache, and also provides an extra 4GB for commodity DRAM capacity.

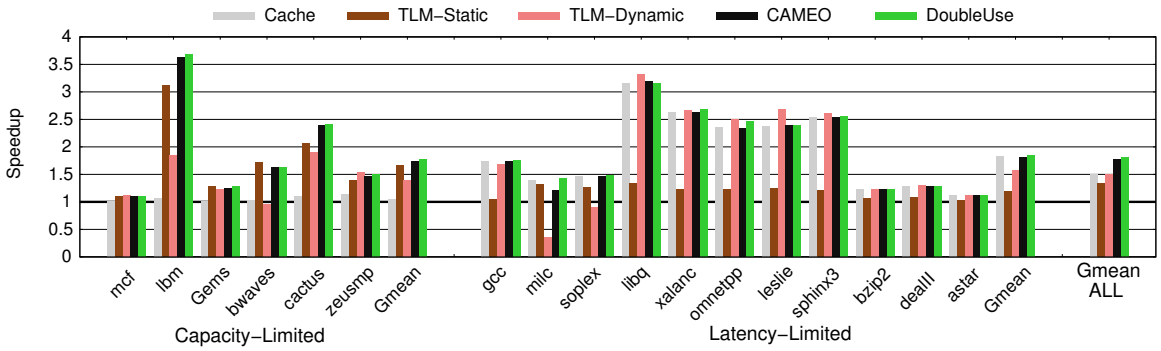

Figure 6.11: Speedup with 3D-DRAM. CAMEO outperforms both cache and two-level memory. CAMEO is close to an idealistic "DoubleUse" design that uses 4GB 3D-DRAM as cache and also increases memory capacity by 4GB (commodity DRAM).

6.5.2 Bandwidth Usage in Memory and Storage

The whole system contains three modules: 3D-DRAM, off-chip commodity DRAM, and storage. An ideal design would reduce the bandwidth consumption of all these three modules simultaneously. However, each of the three designs: cache, TLM, and CAMEO, optimize the bandwidth of different modules. The measurement of bandwidth consumption of different designs is based on the number of bytes transferred on the bus in respective systems and normalized to the number in the baseline. Table 6.4 shows the bandwidth usage of 3D-DRAM, off-chip DRAM, and storage, for different designs, averaged over the workload category. Cache reduces off-chip DRAM banwidth by 45%, However, cache does not reduce storage bandwidth. The reason why cache (and CAMEO) have higher 3D-memory bandwidth usage than the baseline is from installs of data lines. Both TLM-Static and TLM-Dynamic reduce the bandwidth of storage. TLM-Dynamic consumes significant amount of bandwidth for both off-chip DRAM and 3D-DRAM, because of page migration. Thus, TLM-Dynamic optimizes storage bandwidth at the expense of memory bandwidth. CAMEO performs a fine granularity transfer between 3D-DRAM and commodity DRAM, which reduces the memory bandwidth consumption significantly. The 3D-DRAM bandwidth consumption of CAMEO is similar to the design of cache. However, CAMEO does not provide as much savings as cache for off-chip bandwidth as it needs to install lines

evicted from the 3D-DRAM to commodity DRAM. However, unlike cache, CAMEO does provide a storage bandwidth reduction of 21% for capacity-limited workloads.

Table 6.4: Bandwidth usage in DRAM and storage (calculated as bytes transferred, and normalized to baseline).

	Capacity-limited			Latency-limited	
	3D-DRAM	Off-chip	Storage	3D-DRAM	Off-chip
Baseline	n/a	1x	1x	n/a	1x
Cache	1.93x	0.55x	1x	1.76x	0.29x
TLM-Stat	0.26x	0.74x	0.78x	0.25x	0.75x
TLM-Dyn	**2.54x**	**2.19x**	0.78x	1.95x	1.10x
CAMEO	1.89x	1.07x	0.79x	1.51x	0.47x

6.5.3 Energy Analysis

Also, this section analyzes the power consumption and the energy-delay product (EDP) for different designs. The power estimation for commodity DRAM (i.e., DDR3) and storage is derived from the manual of industry-leading products [81, 85, 86], and the 3D-DRAM power is estimated based on a recent report [8]. For capacity-limited workloads, the processor consumes 60% of the power and the rest is split equally between the storage and DRAM. For latency-limited workloads, the processor consumes 70% of the power and DRAM consumes 30%. Figure 6.12 shows the normalized power consumption and energy-delay product (EDP) for various designs. The power consumption increases for all the configurations because of the addition of 3D-DRAM. Overall, cache increases power consumption by 14%, whereas CAMEO by 37%. TLM-Dynamic increases power consumption by 51% because page migration (large granularity) consumes significant power. Cache degrades EDP for capacity-limited workloads because it provides little performance improvement with the addition of 3D-DRAM power. Overall, cache improves EDP by 4%, and TLM-Static improves by 21%, while CAMEO outperforms all designs by providing 49% EDP improvement.

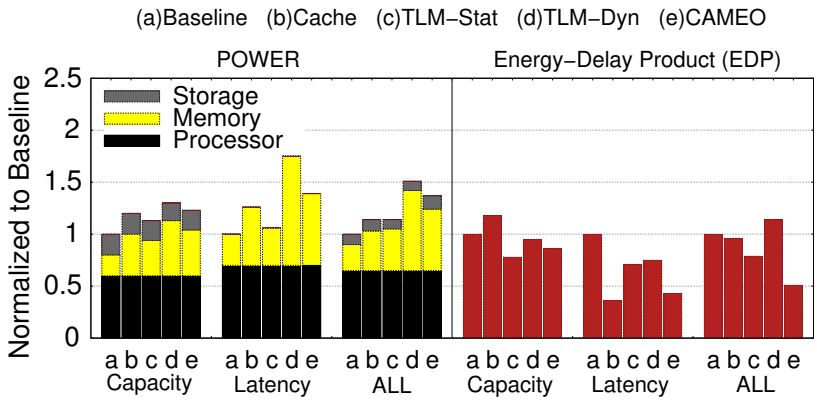

Figure 6.12: Comparison of power and energy-delay product. All the numbers are normalized to the baseline system.

6.5.4 Optimizing Placement for Stacked DRAM

Both CAMEO and TLM-Dynamic migrates recently used data from off-chip DRAM to 3D-DRAM, albeit at a different granularity. The assumption behind the data movement is that the overall performance of the system improves by keeping only the frequently used data in the 3D-DRAM. If the OS has oracular knowledge about page access frequencies, it can place the frequently used pages in 3D-DRAM, and thus avoid the overheads of dynamic page migration. Such an idealistic scheme is referred to as *TLM-Oracle*. Another approach is to track frequency information on page granularity using dedicated hardware and allow the OS periodically perform page migration [76] (referred to as *TLM-Freq*). Note that TLM-Freq requires significant support from both hardware and OS, as memory access frequency is usually not available to the OS at page granularity.

Figure 6.13 compares the speedup of CAMEO with different TLM designs. TLM-Freq ignores the overheads of TLB shootdowns and the software overheads of sorting pages based on access frequencies and performing migration (the bandwidth for page transfer is modeled). For capacity-limited workloads, performing migration at page granularity hurts performance. However, for latency-limited workloads, with small capacity (<4 GB), the page-based scheme ensures all the frequently accessed pages are accommodated in 3D-DRAM. Overall, CAMEO provides 78% performance and TLM-Freq provides 61%.

Therefore, CAMEO outperforms TLM-Freq without the need for page access frequency information and software support for sorting and page migration. Nonetheless, the two optimization techniques are orthogonal and can be combined for further improvement. For example, if page frequency information is available, CAMEO can retain lines from only heavily used pages in 3D-DRAM.

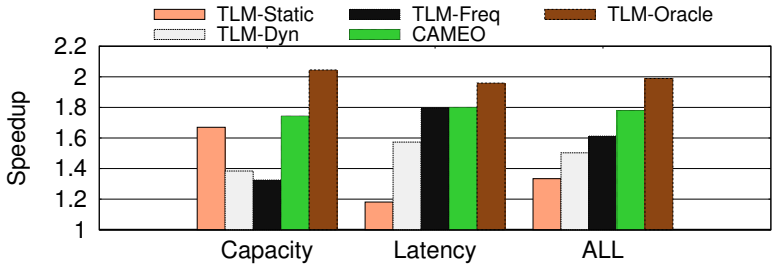

Figure 6.13: Speedup from optimized page placement in TLM. CAMEO outperforms frequency-based page placement without requiring the tracking support.

6.5.5 Sensitivity to Size of Off-Chip Memory

In the baseline parameter of the heterogeneous memory system, 3D-DRAM is one-third the size of off-chip DRAM, which is in line with the future projections of the size of 3D-DRAM [6, 7, 8, 9]. This section analyzes the effectiveness of different designs for various ratio of 3D-DRAM to off-chip DRAM, by using a 4GB 3D-DRAM and varying the size of off-chip DRAM. Figure 6.14 shows the performance of cache, TLM, and CAMEO for different off-chip DRAM size, averaged and normalized to the respective baseline memory system (with no 3D-DRAM). When the relative size of 3D-DRAM is small (1/8 capacity ratio), using 3D-DRAM as a cache is effective, as almost all workloads fit in the off-chip DRAM. For such a system, CAMEO (55%) performs similar to cache (53%), whereas TLM provides negligible benefit as a consequence of the high-bandwidth overheads of page migration. When the size of 3D-DRAM is half of off-chip DRAM, CAMEO outperforms all designs by providing an average speedup of 2.4x, whereas cache, TLM-Static, TLM-Dynamic provide 63%, 89%, and 96%, respectively.

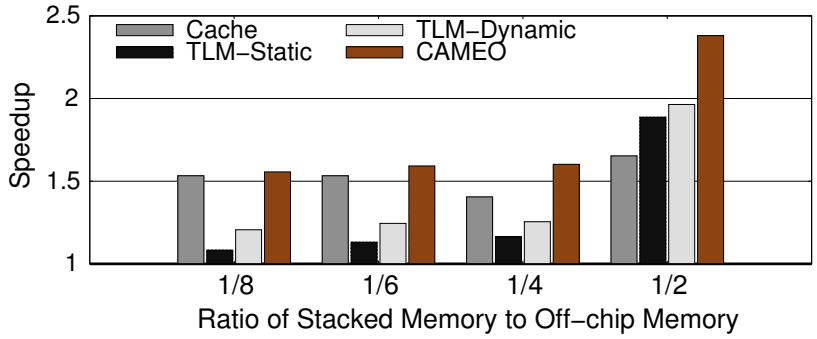

Figure 6.14: Performance impact of varying the ratio of 3D-DRAM (1GB) to off-chip DRAM

6.6 Summary

When the size of 3D-DRAM is a significant fraction of the total DRAM capacity, the effective memory capacity increases if the 3D-DRAM account for OS-visible memory address space. However, being OS-manged memory, the 3D-DRAM requires software support and operates only at the page granularity. This chapter investigates a mechanism that exposes the 3D-DRAM to the OS but still retains the fine-granularity and OS-transparent data migration of caches. To this end, this dissertation proposes a *cache-like memory organization (CAMEO)* that obtains the best of both worlds: main memory and cache. CAMEO exposes the capacity of 3D-DRAM to the OS so that 3D-DRAM counts towards the OS-managed memory address space. Also, CAMEO performs line-granularity data migration transparently, in a manner similar to hardware caches.

For the line-granularity data migration, CAMEO relies on swapping of recently used data lines from off-chip DRAM to 3D-DRAM. As lines can be out of its original place, a request must know the physical address of the line. CAMEO uses a simple and practical *line location table (LLT)* that tracks the physical location of all data lines. To avoid the serialization of LLT look-up and data access, CAMEO co-locates the LLT entry with DATA in 3D-DRAM, which reduces the overhead of data resident in the 3D-DRAM. For data lines resident in the off-chip DRAM, the performance of CAMEO can be improved by removing the serialization latency due to LLT look-ups. CAMEO uses a low-latency (single-cycle),

low-storage overhead (512 bytes), and highly accurate (90%) hardware-based *line location predictor (LLP)* to predict the physical location of a line. The evaluations show that CAMEO provides an average performance improvement of 78%, outperforming alternative design points of hardware cache (50% improvement) and OS-managed two-level memory (33% improvement). The performance of CAMEO is very close to an idealized system that uses 4GB 3D-DRAM as a hardware cache and also increases the off-chip commodity DRAM capacity by an additional 4GB.

CHAPTER 7

DRAM CACHES FOR MULTI-NODE SYSTEMS

This chapter presents the study that scale the heterogeneous memory systems to accommodate multiple 3D-DRAM modules in the system. In particular this dissertation focuses on architecting the DRAM cache for multi-socket systems. Multi-socket systems, which enable large-capacity memory systems, are composed of multiple *sockets* (or *nodes*), each of which has a computing unit (e.g., CPU), a DRAM cache, and also DIMM-based DDR as the main memory. The nodes are connected via a long latency interconnect as shown in Figure 7.1, which is a multi-node system that each node has a 4-core multi-processor, an on-die L3 cache, a DRAM cache, and a DDR-based main memory.

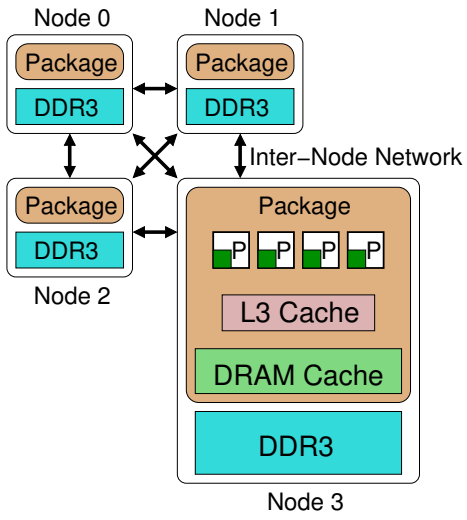

Figure 7.1: Overview of a multi-node System. Each node has a 4-core multi-processor, a shared on-die cache (L3 cache), a DRAM cache, and a DDR-based main memory.

7.1 Problem: Memory-side Cache or Coherent DRAM Cache?

To use DRAM caches in a multi-node system, one practical design is to restrict the DRAM cache in each node to store only the data that belongs to the local node (i.e., *local data*). Figure 7.2(a) shows such design, termed *memory-side cache (MSC)* by the industry vendor [14, 31]. Node 0's DRAM cache holds only the data from node 0 (⬡ and □ symbols in Figure 7.2); similarly, node 1's DRAM cache holds only the data from node 1 (◯ and △ symbols). As any data line is stored in at most one DRAM cache, MSC is implicitly coherent and obviates the need of any coherence support for DRAM caches. However, MSC constraints the system to rely on the small on-die cache for the data from the remote node (i.e., *remote data*). As accessing the remote data incurs long network latency, MSC suffers from a significant latency overhead of on-die cache misses to the remote data.

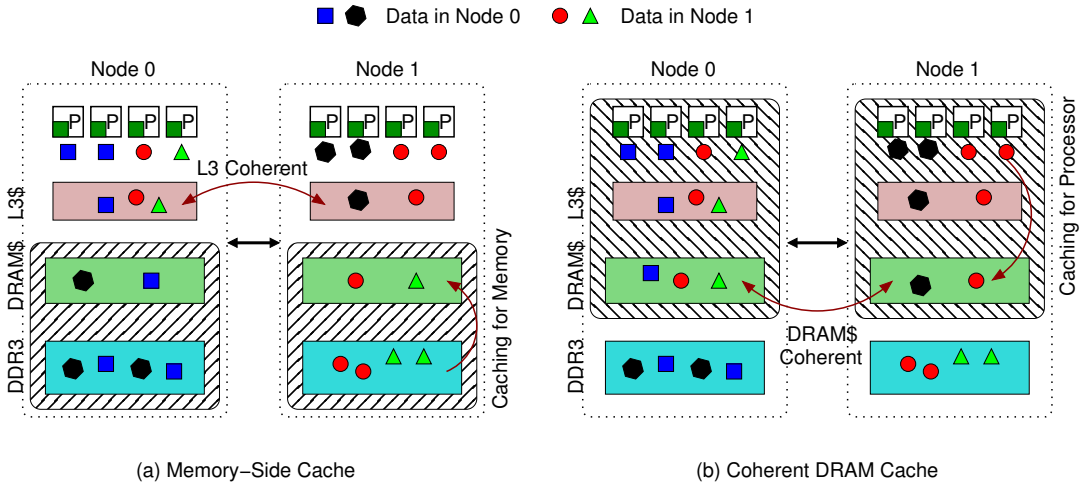

Figure 7.2: Figure (a) and (b) show different usage of DRAM caches in multi-node systems. Each symbol in (a) and (b) represents a data block in the memory. (a) Memory-Side Cache. The DRAM cache in node 0 is allowed to cache only the data that is in Node 0 (□ and ⬡ in this case); same for Node 1 (◯ and △). On-chip L3 caches must still be be kept coherent. (b) Coherent DRAM Cache. The DRAM cache stores data from both nodes. Node 0's DRAM cache stores data blocks requested by node 0 and caches data from both Node 0 and Node 1 (□, ◯ and △); same for Node 1 (⬡ , and ◯). In this case, DRAM caches must be kept coherent.

A desirable alternative is to allow the DRAM cache to cache both the local and the remote data to mitigate the long network latency overhead. Given that the DRAM cache capacity is in the range of gigabytes, it is capable of holding a much larger working set, hence reducing the needs to access remote nodes. Figure 7.2(c) shows such design: node 0's DRAM cache holds all request initiated from processors in node 0 and caches the data from both node 0 and node 1 (□ from node 0 and ◯ and △ from node 1); node 1's DRAM cache does the same for requests initiated by the processors in node 1 (⬡ from node 0 and ◯ from node 1). As this type of DRAM cache stores data blocks from any node, a shared data block can be stored in multiple DRAM caches (e.g., ◯ stored in DRAM caches of both node 0 and node 1). Therefore, the DRAM caches must be kept coherent for correctness, and such a design is referred to as *coherent DRAM cache (CDC)*. In this case, the DRAM cache becomes the last-level cache and the point of cache coherence.

7.1.1 Directory-based Coherence Protocol

Current multi-node systems typically use directory-based coherence protocol because of its superior scalability [37, 38, 39]. One appealing design is to use *sparse directory* to track the data blocks that are currently being cached (and need to be coherent) in the system. Commercial products implement sparse directory by dedicating part of the die area. For example, AMD's Magny-Cours has 1MB SRAM structure (referred to as *Probe Filter* by AMD) for the sparse directory [87, 88]. As sparse directory stores coherence information, such a structure is referred to as the *coherence directory (CDir)* [1], which is distinguish from the *tag directory (TDir)*, which stores tag information for data blocks in the cache. Figure 7.3 shows the organization of CDir and TDir. The TDir, associated with one coherent cache, has the state information (e.g., valid, dirty) for all lines that are currently stored in its associated cache. For example, in node 0, processors request for three data blocks (□, ◯, and △). Therefore, the TDir in node 0 has information of these three lines.

[1] Sparse directory, also referred to as *directory cache*, or *standalone inclusive directory with recall* in other literatures [43, 44, 45, 89, 42], is not to be confused with full coherence directory [37].

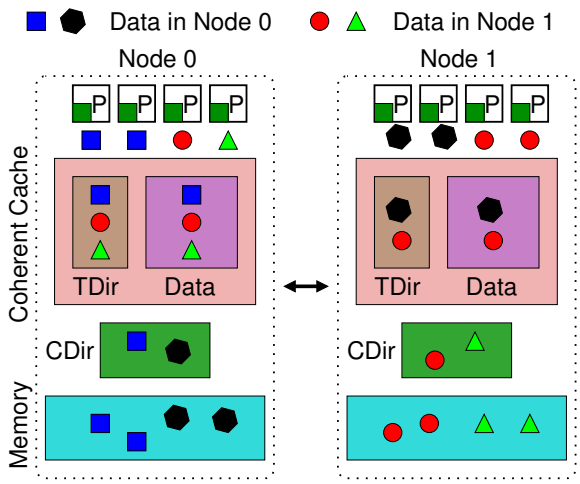

Figure 7.3: Overview of tag directory (TDir) for cache and coherence directory (CDir) for data. Notice that they are responsible for different data blocks that are currently being cached in the system.

Unlike the TDir that stores state information of lines in the cache, the CDir, associated with the memory in a node, has the coherence information (e.g., shared, exclusive, etc., depending on the coherence protocol) for all the cache blocks that belong to its memory and are currently being cached (by any node) in the system. For example, two blocks (△ and ◯) from node 1 are currently being cached; the CDir in node 1 keeps information of these two lines. Node 1 is referred to as the *home node* for these two lines. In directory-based protocol, the home node is responsible for retrieving the most recent copy of the data on a L4 cache miss: First, the home node accesses the CDir for the coherence information of the requested data block. Based on the request type of the cache miss and the coherence state of the requested data block, the home node takes different operations. For instance, if the requested data block is uncached, the home node accesses the memory to retrieve the data. Other operations include invalidating a copy in others' caches or requesting the owner to write back the most recent data. To apply the directory-based coherence protocol to CDC, the dissertation identifies two key problems, described in the following sections.

7.1.2 The Need for Large Coherence Directory

The directory-based coherence protocol requires a CDir for the L4 DRAM cache. This section first examines whether the current architecture that keeps on-die L3 cache coherent can be applied to the L4 caches.

On-Die L3 Coherence Directory

In the baseline MSC, an on-die L3 coherence directory, referred to as the *OnDie-CDir*, is responsible for maintaining L3 cache coherence. For example, the aforementioned probe filter by AMD. In CDC, the point of coherency is the L4 cache; one simple way to organize the CDir for L4 caches is to use the same OnDie-CDir. Although this approach reuses the existing resources and does not incur extra overhead, using OnDie-CDir as the L4 coherence directory, compared to a baseline memory-side cache, degrades performance by an average of 24%, with a maximum of 66% (detailed methodology in Section 7.4). Therefore, using existing OnDie-CDir for DRAM caches jeopardizes the use of CDC. The performance degradation stems from the insufficient coverage of OnDie-CDir, as the explanation follows in the next paragraphs.

Coverage of Coherence Directory

A CDir entry includes the memory address, the state (e.g., modified, exclusive, etc.), and a sharer bit vector for sharers or the owner. Every cached block must have a corresponding entry in the CDir; when a valid CDir entry is replaced, it invalidates the corresponding cache block in L4. Such invalidation is referred to as *coherence-induced invalidation*. To minimize coherence-induced invalidation, the number of CDir entries must be proportional to the cache capacity. The ratio of the number of the entries in CDir to the number of cache blocks in the cache is referred to as *coverage* of the coherence directory. An 1X-coverage CDir, with as many entries as the number of cache blocks in the system, is the minimum coverage so that DRAM cache capacity is fully used. Prior studies suggest that

CDir must have at least 2X coverage to minimize coherence-induced invalidations. For examle, AMD's Magny-Cours provisions a 1MB CDir, which can cover a total of 16MB (at 4B per CDir-entry), whereas the system contains a total of 8MB cache capacity (eight 256KB L2 cache and one 6MB L3 exclusive cache) [87, 88].

Size of Coherence Directory

The size of the coherence directory depends the coverage requirement for a 1GB DRAM cache, which holds 16 million cache blocks. Figure 7.4 varies the coverage and shows the corresponding DRAM cache miss rate. The OnDie-CDir, with 256K entries or an equivalent coverage of $\frac{1}{128}$X, limits the effective capacity of L4 caches; therefore, it results in very high DRAM cache miss rate and degrades performance. For an 1X coverage, the CDir needs 16 million entries for a 1GB DRAM cache (16 million blocks). Even each entry is only 4 bytes, the size of the coherence directory would be 64MB.[2] Note that the required coverage is a function of application's working set and access behavior. While 1X coverage seems sufficient, certain application, suggested by prior studies [42, 43, 44, 45], could require higher coverage (at even higher storage cost). The study assumes assumes a 1X coverage (64MB) and discusses the extension for a 2X coverage (128MB).

7.1.3 The Need for Low-Latency Request-For-Data Operation

The other challenge to architect giga-scale CDC is a low-latency *request-for-data (RFD)* operation. The request-for-data (also referred to as *Fwd-GetS* in other literature [89]) is the operation that the home node asks the owner to write back the most recent copy of the data. Therefore, the request-for-data operation reads the most recent data via a cache access, and thus is on the critical path. Figure 7.5 that follows Figure 7.3 illustrates an example. The black circle represents the sequence of the operations. Now consider a read miss to a

[2]Given 16 million entries, each CDir entry requires 22-bit tags (48-bit physical address, 6-bit line offset, 20-bit set indexing for 16-way associativity), 1 valid bit, 2 state bits, and 4 bits for the sharer vector, so the size of each CDir entry is 29 bits. One CDir entry provisions 32 bits (4 Byte).

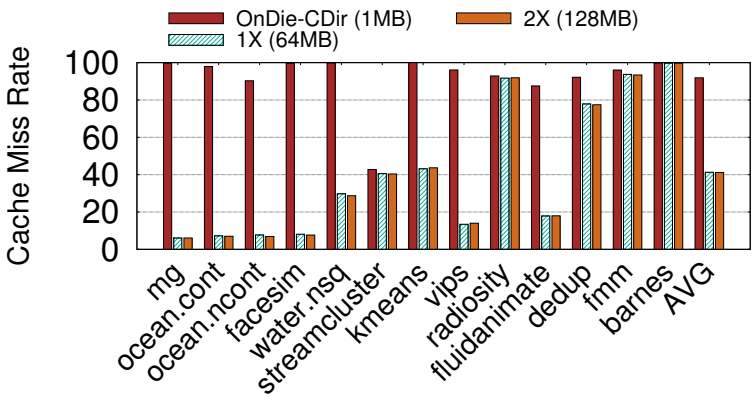

Figure 7.4: Impact of Coherence Directory Coverage on DRAM Cache Miss Rate. This study uses a 1GB DRAM cache, and shows OnDie-CDir (equivalent to $\frac{1}{128}$X) and coverage of 1X and 2X.

modified data block (⬡) ❶. The CDir in the home node (node 0) indicates that node 1 is the owner of the block ❷. The home node requests node 1 to write back the most recent data via a request-for-data operation ❸. When the owner receives the request-for-data, it reads its copy of the data from the cache ❹ and replies to the home node. After receiving the data, the home node then replies to the requester and makes the coherence state *shared*.

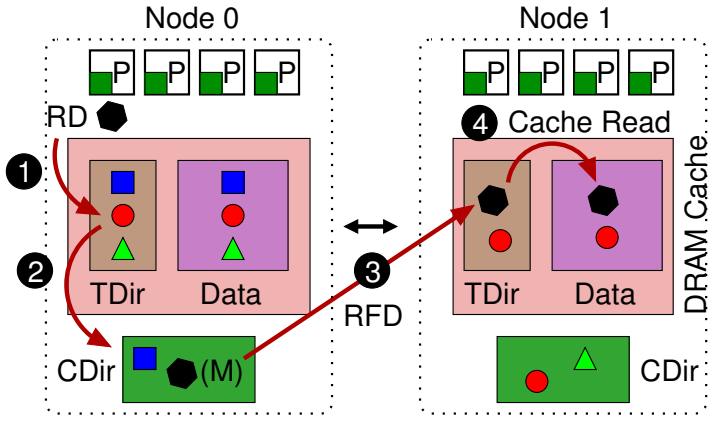

Figure 7.5: Sequence for a Request-For-Data (RFD) Operation

Figure 7.6 shows a latency breakdown of the request-for-data operation that reads the data in the cache for both MSC and CDC. Figure 7.6(a) shows the case of MSC, where L3 caches are the point of coherency. Therefore, the cache access latency of request-for-data is L3 cache latency. The total latency of request-for-data is L3 cache latency plus

the round-trip interconnect latency. In contrast, the cache access latency becomes DRAM cache latency when it comes to CDC in Figure 7.6(b), because L4 caches are the point of coherency. The total latency of request-for-data is L4 cache latency plus the round-trip interconnect latency. The L3 cache latency is 8ns, the L4 cache latency is about 40ns, while the round-trip latency is about 50ns [90, 91]. Therefore, the request-for-data operation in CDC incurs a significantly long latency.

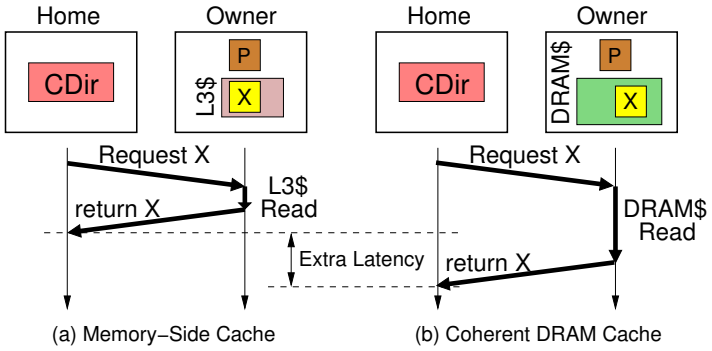

Figure 7.6: Latency breakdown of request-for-data operation in memory-side cache (coherent L3), and coherent DRAM cache (coherent L4). Latency not to scale.

7.1.4 Performance Potential of CDC

Although two challenges of architecting giga-scale CDC prevents applying existing architecture, the key question is how CDC performs. To understand the performance potential of CDC, Figure 7.7 shows the performance improvement by CDC with respect to MSC. The CDC is idealized by an impractical SRAM-based coherence directory of an 64MB SRAM overhead and zero L4 DRAM cache access latency for request-for-data operations (still incurs inter-node network latency). Such an idealized design is referred to as *Impractical-CDC*. On average, the Impractical-CDC outperforms MSC by 30%, with a maximum speedup of 2.8X from *ocean.cont*. Note that workloads with significant performance improvement tend to have large footprint of private data or read-only shared data (detailed workloads in Section 7.4 and Table 7.3). Therefore, for such workloads, CDC avoids inter-node network latency and significantly improves performance.

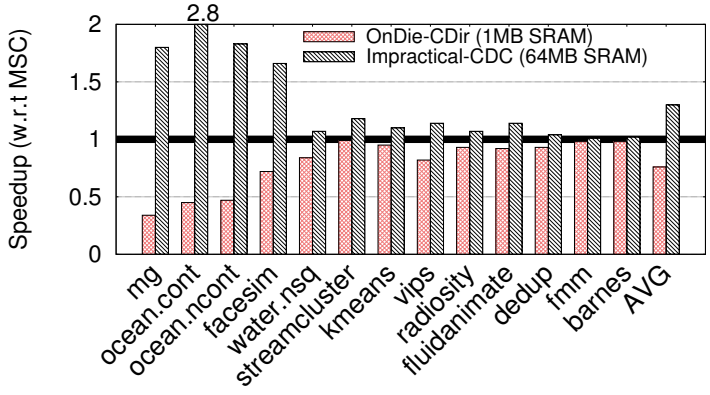

Figure 7.7: Performance of CDC using OnDie-CDir and impractical coherent DRAM cache (Impractical-CDC)

7.2 DRAM-cache Coherence Buffer: A Low-latency Coherence Directory

To architect high-performing giga-scale Coherence DRAM Cache, the dissertation proposes *DRAM caches for multi-node systems* (*CANDY*), which has two orthogonal components to address the challenges. This section (Section 7.2) investigates the coherence directory for giga-scale L4 caches and leverages existing resources for a low-latency coherence directory, while Section 7.3 provides further analysis on the request-For-data problem and proposes a technique to mitigate the latency by exploiting the characteristics of read-write shared data.

7.2.1 Coherence Directory Organization

SRAM-Based Coherence Directory

One way to build a coherence directory is to use a separate storage structure to keep the coherency information [42, 43, 44, 45, 87, 88]. The same principle suggests two simple way for the coherence directory of giga-scale CDC. The first approach is to use SRAM storage. However, given the size of the coherence directory (64MB, larger than L3 cache), putting it on die is prohibitively expensive. Figure 7.8(a) shows the design of SRAM-based CDir, referred to as the *SRAM-CDir*. In SRAM-CDir, the latency to access the CDir is the SRAM access latency.

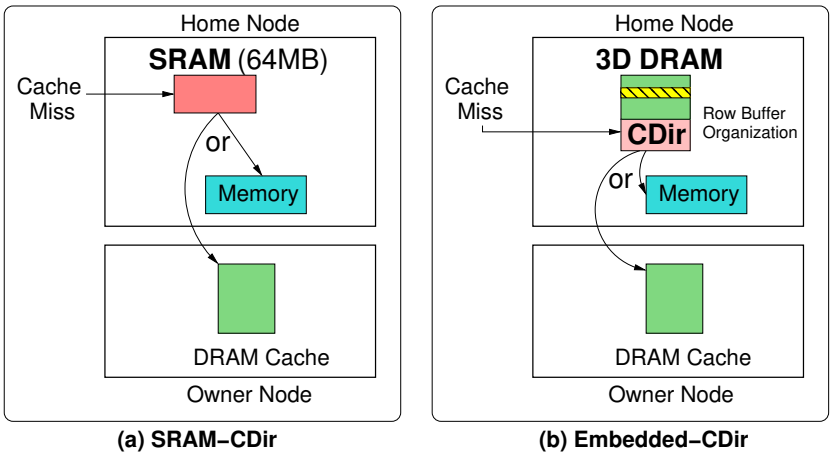

Figure 7.8: Coherence Directory Organization: (a) SRAM-based coherence directory and (b) embedded coherence directory.

Embedding Coherence Directory in 3D-DRAM

Alternatively, a practical design to accommodate such a large structure is to place the coherence directory in the 3D-DRAM to avoid SRAM storage overhead. A portion of the 3D-DRAM capacity is reserved for the CDir, thus reducing the DRAM cache capacity. This embedded approach is referred to as *Embedded-CDir*, shown in Figure 7.8(b). Figure 7.9(c) illustrates how to access the DRAM cache and the Embedded-CDir in 3D-DRAM and shows an example of the state-of-the-art alloy cache [30]. In DRAM caches, each cache access reads a basic access unit of 72 bytes (8B TDir and 64B Data). For simplicity, the Embedded-CDir also uses the same basic access unit. That is, one access to the Embedded-CDir also reads 72 bytes, which includes 18 CDir entries (4B each). Note that the use of alloy cache is only for illustration purpose; the idea that embeds the coherence directory in 3D-DRAM can be applied to other DRAM cache designs.

The organization of the Embedded-CDir, such as set associativity, determines the effectiveness of reducing coherence-induce invalidations. For example, an 18-way set-associative structure (referred to as *high-assoc*), given each access returns 18 CDir entries. Figure 7.10 shows the performance for Embedded-CDir that is high-assoc, and low-assoc

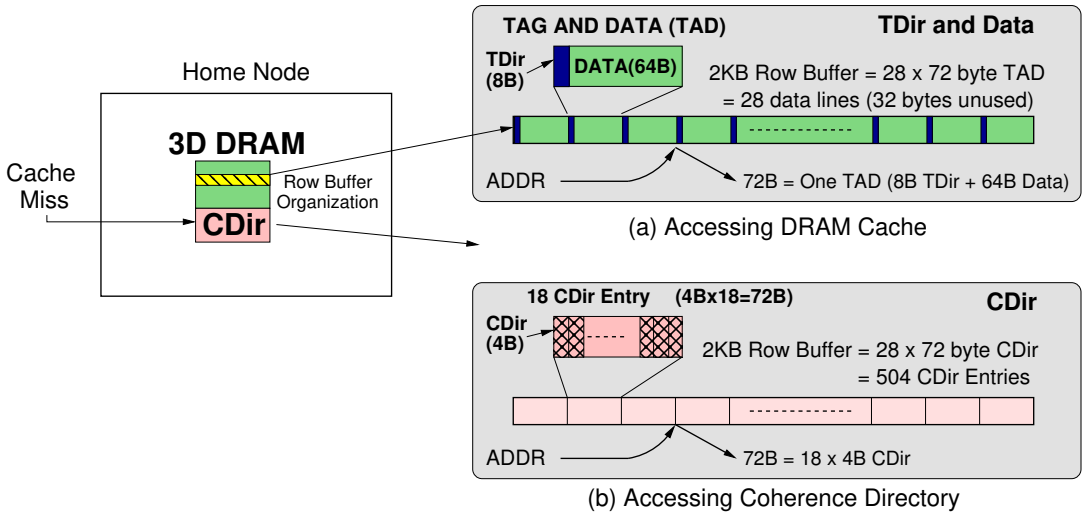

Figure 7.9: (a) Cache access and (b) Embedded-CDir access: One read access to Embedded-CDir in 3D-DRAM gets 18 CDir entries (72 bytes). Note that the figure uses alloy cache [30] as an example.

(direct-mapped, 18 sets for 18 CDir entries). On average, high-assoc improves performance by 11%, while Low-Assoc is 10%. Although high-assoc delivers the higher performance, later section will discuss how lower set-associativity furthers improve performance. In addition, Figure 7.10 also shows the performance improvement by SRAM-CDir. On average, SRAM-CDir improves performance by 25%, while Embedded-CDir has only 11%. Moreover, Embedded-CDir degrades performance for several benchmarks, such as *streamcluster* (42%) and *barnes* (15%). Therefore, although Embedded-CDir is a more practical design to reduce the costly SRAM storage overhead, it does not perform as well as the SRAM-CDir, because of the latency overhead of accessing the Embedded-CDir in 3D-DRAM.

7.2.2 Leveraging On-Die Coherence Directory

While the SRAM-CDir design has fast access latency, the Embedded-CDir design has a lightweight overhead. An ideal case is to have the best of both worlds. To this end, this dissertation proposes to re-purpose the on-die SRAM coherence directory (OnDie-CDir, meant for L3 cache coherence in MSC) as a buffer to store the recently accessed CDir

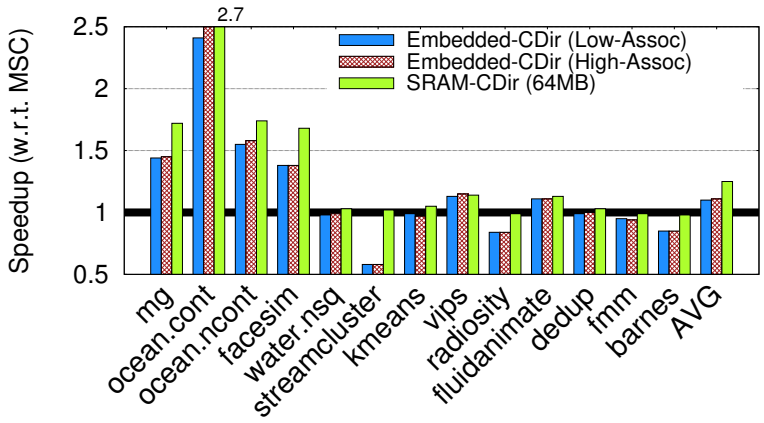

Figure 7.10: Performance of Embedded-CDir (low-Assoc and high-Assoc) and SRAM-CDir.

entries from Embedded-CDir. Recall that OnDie-CDir is provisioned and used for the L3 cache coherence in MSC (described in Section 7.1.2); in CDC, as L4 cache becomes the point of cache coherence, the OnDie-CDir is unused. The idea is to leverages such an existing SRAM structure in CDC. The re-purposed structure is referred to as *DRAM-cache Coherence Buffer* (*DCB*). Figure 7.11 shows the overview of DCB and also its interaction with Embedded-CDir. On a cache miss, the home node first checks the DCB to find the corresponding entry. If the entry is found in DCB (a *hit* in DCB), the latency to access the CDir entry is only SRAM access latency. On the other hand, if the entry is not found in DCB (a *miss* in DCB), the home node retrieves the entry from Embedded-CDir by issuing a 3D-DRAM read access. In this case, the latency to access the CDir entry is the sum of the DCB latency and the Embedded-CDir latency. After Embedded-CDir returns the entry, the home node inserts the entry into DCB for future references.

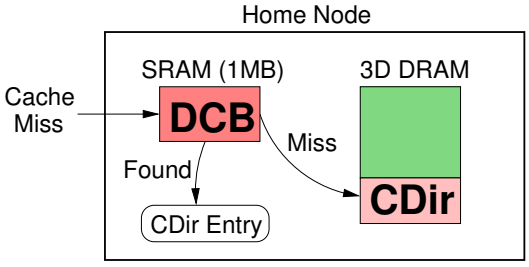

Figure 7.11: Overview of DRAM-cache Coherence Buffer (DCB) and Embedded-CDir. On a cache miss, the home node first checks the DCB; if the entry misses in DCB, the home node checks the Embedded-CDir in 3D-DRAM.

7.2.3 Design of DRAM-cache Coherence Buffer

The latency to retrieve the CDir entry is determined by whether the entry is found in DRAM-cache coherence buffer, or, in other words, the *hit rate* of DCB. Higher DCB hit rate leads to lower average latency. Therefore, the effectiveness of mitigating CDir access latency depends on the hit rate of DCB. The hit rate of DCB is a function of its size and its interaction with the Embedded-CDir, and this section presents two simple designs for DCB to maximize the hit rate.

Exploiting Temporal Locality

After a DCB miss, the home node retrieves the corresponding CDir entry by accessing the Embedded-CDir and performing tag-matching. To exploit temporal locality, the home node inserts the demand missing CDir entry into DCB so that future cache misses are likely to hit the same entry in the DCB. As this design inserts CDir entries into DCB on demand misses, it is referred to as *DCB-Demand*. To minimize the misses of DCB, the DCB maximizes the set associativity; therefore, the DCB is organized as a 16-way set-associative structure, indexed by the memory address.

Exploiting Spatial Locality

Besides temporal locality, the second design is to exploit spatial locality to improve the DCB hit rate. For example, for a streaming workload that sequentially accesses the memory, a cache miss to memory address X implies a high likelihood that the subsequent cache miss would go to memory address close to X [92, 93, 94]. In the context of CDir entries, it means that the next requested coherence directory entry is spatially correlated to the currently requested CDir entry. This suggest that the DCB and the Embedded-CDir can be organized for spatial locality by exploiting the access granularity of 3D-DRAM.

To exploit spatial locality, the DCB must fetch multiple CDir entries (across sets) and install both demand and spatially correlated CDir entries in the DCB. In high-assoc

Embedded-CDir, organized as a 18-way set-associative structure, one 3D-DRAM access returns only one set; fetching additional sets incurs extra 3D-DRAM read. One way to avoid such bandwidth overhead is to use lower set-associative Embedded-CDir such that the CDir entries of continuous memory addresses are placed in one 3D-DRAM access unit and fetched in one 3D-DRAM access. Figure 7.12 shows an example that one 3D DRAM access returns 4 sets (medium set associativity). (One access gives 18 entries, so DCB uses 5-way for odd sets and 4-way for even sets.) One 3D-DRAM access fetches CDir entries for up to 4 continuous memory addresses from different sets. Therefore, every access to the Embedded-CDir returns not only the requested entry but also entries of the continuous addresses. In Figure 7.12, the address finds an extra match and inserts one additional CDir entry (address *X+2*). This design is referred to as *DCB-SpaLoc*.

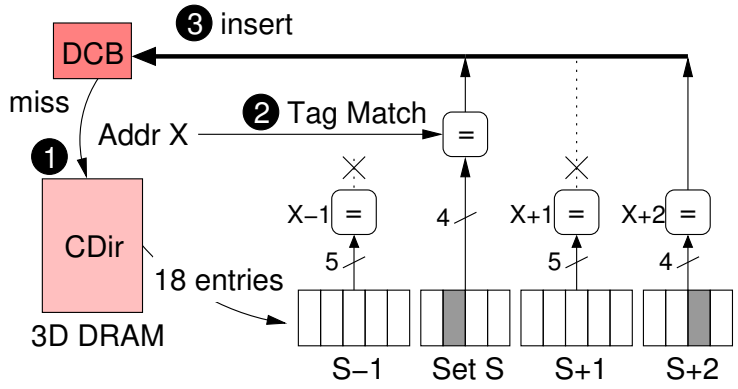

Figure 7.12: Optimizing DCB and Embedded-CDir for Spatial Locality.

7.2.4 Effectiveness of DCB

DCB Hit Rate

Figure 7.13 shows the hit rate of DCB for both DCB-Demand and DCB-SpaLoc designs. Recall that DCB is re-purposed from OnDie-CDir, which has a fixed area budget of 1MB SRAM. In such allowance, DCB-Demand has an average DCB hit rate of 75%, while DCB-SpaLoc has an average DCB hit rate of 81%.

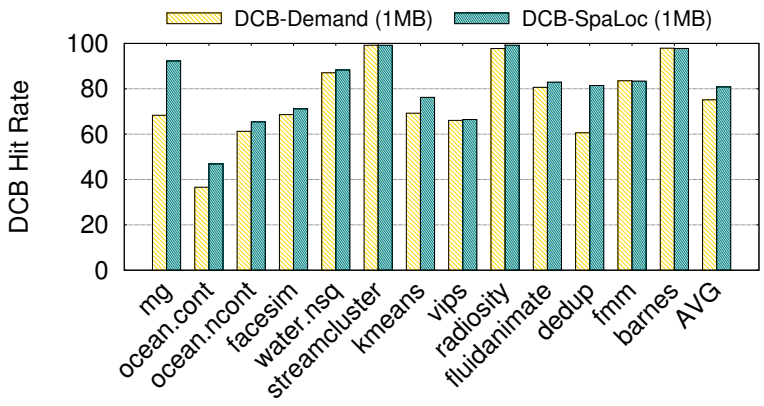

Figure 7.13: DCB Hit Rate: DCB-Demand and DCB-SpaLoc. Both are allowed for 1MB SRAM in size.

Performance

Figure 7.14 compares the performance for the Embedded-CDir, DCB-Demand, and DCB-SpaLoc, as well as a case in which DCB is perfect with 100% hit rate (referred to as *DCB-Perfect*). On average, DCB-Demand outperforms the baseline MSC by 18%, while DCB-SpaLoc improve performance by 21%, and DCB-Perfect has a performance improvement of 25%. DCB-Demand and DCB-SpaLoc mitigate the performance degradation introduced by the Embedded-CDir (e.g., mg, streamcluster, and radiosity), and outperform the Embedded-CDir by 7% and 10%, respectively. Also, the improvement of DCB hit rate by DCB-SpaLoc reflects on the performance improvement (3% over DCB-Demand).

7.3 Sharing-Aware Bypass: Architecting Low-Latency Request-For-Data

Request-for-data is a critical and necessary operation that reads the most recent data from a cache. Unfortunately, in CDC, such an operation incurs a 3D-DRAM access latency, which is much higher than the latency in its counterpart MSC. This section investigates the request-for-data problem in CDC and proposes a technique to mitigate the cache access latency of request-for-data by exploiting the characteristics of read-write shared data.

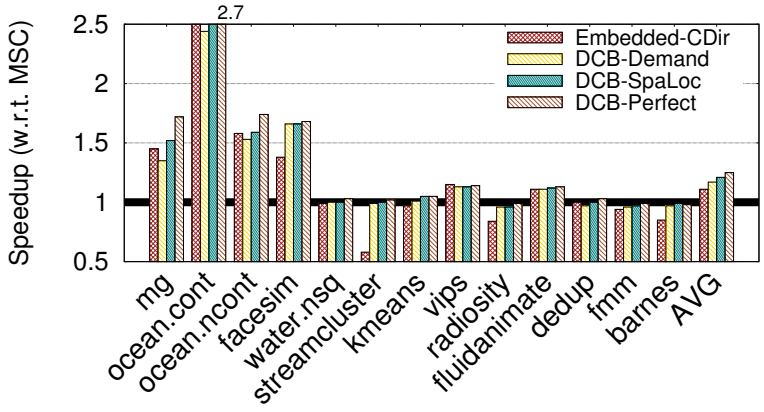

Figure 7.14: Performance Comparison of Embedded-CDir, DCB-Demand, DCB-SpaLoc and DCB-Perfect

7.3.1 Request-For-Data: What and Why?

To maintain coherent caches, coherence protocol relies on different operations based on the type of the request and the coherence state of the requesting block. Such operations include accessing the memory and *coherence operations*. While the memory access fetches data from the memory, the coherence operations are dedicated to keeping the cached data up-to-date. Without loss of generality, coherence operations are classified into three categories: (1) *request-for-data* (*RFD*), which asks the owner to write back the most recent data; (2) *invalidation* (*INV*), which invalidates the copy in a cache; and (3) *flush*, which is a combination of RFD and INV) [38, 39]. Note that request-for-data is referred to as *Fwd-GetS*, and flush is referred to as *Fwd-GetM* in MESI protocol [89]. Although the study uses MESI as an example, specific coherence operations depend on the coherence protocol, but all protocol require similar operations to maintain coherence. For MESI protocol, Table 7.1 shows the detailed classification.

An INV updates the TDir (valid bit) by a write access to TDir, while an RFD and a flush must access the data in the cache via a read access to the cache. In CDC, the RFD operation reads the data from the DRAM cache and incurs a 3D-DRAM read access. Therefore, the RFD latency in CDC is much longer than in MSC, where RFD incurs only a L3 SRAM

Table 7.1: Classification of coherence operations based on request type and the coherence state of the data block

State	Read	Exclusive/Write
Modified	Request-for-data	Flush
Exclusive	Request-for-data	Flush
Shared	Memory Read	Invalidation
Invalid	Memory Read	Memory Read

cache access latency (Illustration shown in Figure 7.6). Moreover, the RFD is on the critical path, and such an RFD latency overhead in CDC can penalize the performance. However, RFD is a critical problem only if it accounts for significant percentage of the operations. Figure 7.15 shows the percentage of four operations with respect to DRAM cache accesses (i.e., L3 cache misses): request-for-data (RFD), invalidate (INV), memory (Mem), and cache hit. (flush counted as RFD, as a flush includes an RFD.) On average, the L4 cache hit rate is 59%; RFD contributes to 20% of the L3 cache misses accesses, while 12%, and 9% of the L3 misses are INV and Mem, respectively. Therefore, as 49% of the DRAM cache misses result in RFD operations, mitigating the latency for RFD is a key challenge of a high-performance CDC.

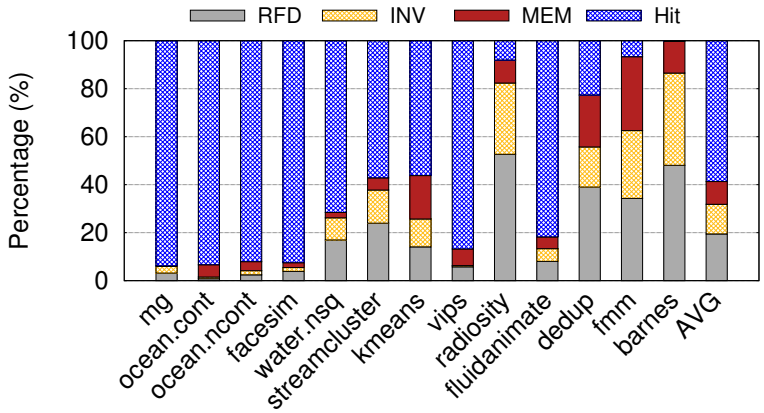

Figure 7.15: Percentage of DRAM cache requests: request-for-data (RFD), invalidate (INV), memory (MEM) and cache hits.

7.3.2 Bypass, Don't Cache (in DRAM caches)

Although RFD reads data from a cache, not all data in the cache are accessed by RFD operations: As RFD maintains cache coherence for *read-write shared data*, only read-write shared data are accessed by RFD [51]. Therefore, if such data are not read from the L4 DRAM caches, but from the L3 caches, the cache access latency of RFD reduces significantly from DRAM cache latency to L3 cache latency. One simple way to achieve this goal is to use select caching or cache bypassing [95]. In other words, the read-write shared data *bypass* L4 caches and are stored only in L3 caches. To this end, the dissertation proposes *sharing-aware bypass* (*SAB*), which enforces read-write shared data to bypass L4 caches at run-time. Figure 7.16 shows the overview of sharing-aware bypass. SAB is composed of two parts: dynamically detecting read-write shared data, and enforcing the bypassing policy at L4 DRAM caches.

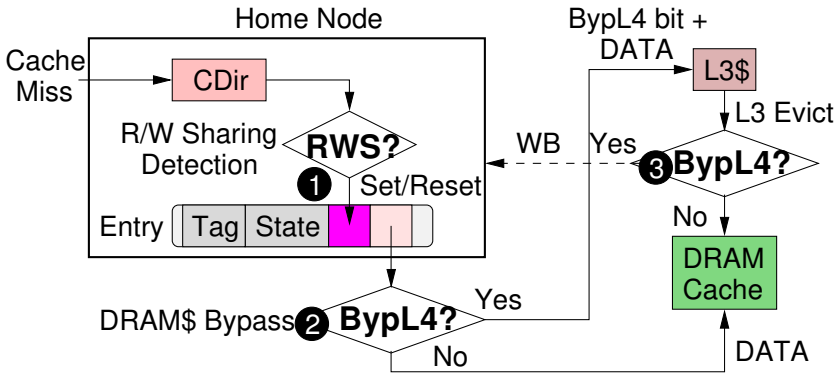

Figure 7.16: Overview of Sharing-Aware Bypass: (1) read-write shared data detection, (2) cache miss and DRAM cache bypass, and (3) L3 dirty eviction and bypassing DRAM Cache.

7.3.3 Detecting Read-write Shared Data

Detecting read-write shared data requires a mechanism that is based on the events of coherence operation [43, 45]. The read-write shared data is the necessary and sufficient condition for coherence operations, so read-write shared data can be easily identified when a given re-

quest incurs such operations. (otherwise, it is read-only shared data or private data.) Thus, on the event of any coherence operations, a data block is classified into read-write shared data or the other. On INV, RFD, and flush operations, the data block is marked *read-write shared*, which requires one bit in the CDir entry to keep track of this classification. On a memory read operation, the bit is reset to mark the data as not read-write shared data. The bit is referred to as *read-write shared (RWS)* bit. As a data block can transition into one mode from the other, the detection mechanism dynamically identifies read-write shared data at run-time, as shown in Figure 7.16.

7.3.4 Enforcing Sharing-Aware Bypass

To enforce the bypassing policy for respective types of data, SAB must determine the bypassing decision and the enforcement of the bypassing decision in the system.

When to Decide

Once SAB identifies a read-write shared data, it can use such information to decide the bypassing policy. The bypassing decision is determined only on the event of ownership change, because ownership change implies that only one node, the requesting node, in the system will have the copy of the data. This avoids unnecessary invalidation to memory address that is shared by multiple sharers in the system. The bypassing decision is stored in the CDir entry associated with the data block by using another bit in the CDir entry. This bit is referred to as *Bypass L4 (BypL4)* bit. For simplicity, SAB decides to use bypassing for every read-write shared data block; that is, if the RWS bit is set, SAB also sets the BypL4 bit.

How to Enforce

Once the decision of a block is determined, SAB must enforce the decision. SAB maintains a uniform bypassing decision for all nodes in the system. This means if one data block

bypasses DRAM caches, such a block cannot be stored in any DRAM caches in any case. However, a data block would attempt to be stored in DRAM caches in two cases: (1) **a L4 Cache Miss** (to be installed to L4 caches). A L4 cache miss goes to the home node to request the data. When replying to the requester, the home node communicates the BypL4 bit in the message with data. The requesting node examines the BypL4 bit, and the data bypass the DRAM cache if the BypL4 bit is set. (2) **a L3 dirty Eviction** (to be written back to L4 caches). On a L3 dirty eviction that attempts to write the data to DRAM caches, the L4 DRAM cache must know the bypassing decision. L4 caches can choose to consult the home node for such information; however, this significantly and unnecessarily increases the latency as well as the network traffic. Alternatively, SAB stores the bypassing information (i.e., the BypL4 bit) in the L3 cache by adding one bit in the tag directory of L3 cache. A L3 dirty eviction uses the BypL4 bit to decide whether the evicted data block should bypass the L4 cache. If the bit is set, meaning bypassing L4 caches, the L3 cache writes the modified data back to the home node.

7.3.5 Analysis of Sharing-Aware Bypass

The effectiveness of SAB depends on the number of RFD serviced by the L3 caches. Compared to a bypassing scheme that has the oracle knowledge of the read-write shared data and uses L3 caches for all the RFD, SAB uses L3 cache for 78% of the RFD. The effectiveness of SAB reflects on the performance improvement, shown in Figure 7.17. On average, SAB provides a speedup of 4% in addition to the improvement of DCB, and overall outperforms MSC by 25%. The storage overhead of sharing-aware bypass includes two bits in the CDir entry, and one bit per line in L3 caches. As the CDir entry is provisioned to be 32 bits (4 bytes), and only 29 bits are used, two bits in CDir entry do not incur any storage overhead. For a 4MB L3 cache (64K lines), one bit per line incurs an SRAM overhead of 8KB per node. Therefore, sharing-aware bypass incurs negligible storage overhead (less than 0.2% L3 area).

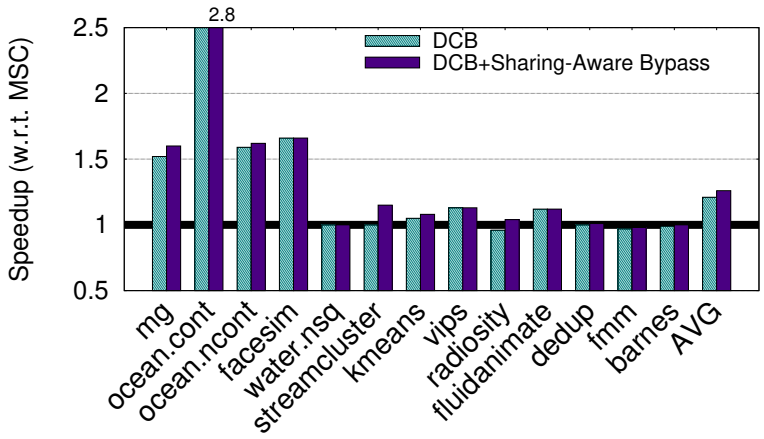

Figure 7.17: Performance of Sharing-Aware Bypass

7.4 Methodology

7.4.1 System Configuration

The experiments are conducted on Sniper [96] simulator. The configuration of the system parameter is shown in Table 7.2: The experiments evaluate a 4-node system, where each node has 4 processors, a shared L3 on-die cache, a DRAM cache, and also DDR3 memory. The timing and bandwidth specification of DRAM cache are modeled after high-bandwidth memory [9]. Each processor has private L1-D, L1-I, and L2 caches. Inter-node communication relies on high-speed links, modeled after Intel's QPI and AMD's Hyper-Transport [90, 91]. The DRAM caches in both memory-side cache and coherent DRAM cache are based on alloy cache [30], which is equipped with hit-miss predictor [67, 30]. MSC is the default baseline system, unless stated otherwise, and the reported performance numbers are normalized to MSC.‘ The default coherence protocol is MESI coherence protocol [97]. The coherence directory (CDir) is distributed and associated with each node. Each entry in the CDir uses a full sharer vector to record the sharing information. For MSC, the coherence directory for L3 cache coherence is 1MB and located on-die. It tracks 256K cache blocks, same as AMD's Magny-Cours [87].

Table 7.2: System Configuration for Multi-socket System Study

Node		DRAM Cache	
Number of Nodes	4	Capacity	1GB
Each Node Configuration		Bus Frequency	800MHz (DDR 1.6GHz)
Processors		Channels	8 64-bit bus
Number of Cores	4	Banks	16 Banks per rank
Frequency	3.2GHz	Row Buffer Size	2048 Bytes
Last Level Cache		tCAS-tRCD-tRP-tRAS	11-11-11-45 bus cycle
Shared L3 Cache	4MB, 16-way, 24 cycles	Main Memory (DDR-based DRAM)	
Coherence Protocol		Capacity	16GB
Protocol	MESI	Bus Frequency	800MHz (DDR 1.6GHz)
On-Die L3 Directory	1MB	Channels	2 64-bit bus
Inter-node Network		Banks	8 Banks per rank
Bandwidth	12.4GB/s	Row Buffer Size	2048 Bytes
Latency	50 ns, one-way	tCAS-tRCD-tRP-tRAS	11-11-11-45 bus cycle

7.4.2 Workloads

The workloads are parallel multi-threaded programs from various benchmarks suites, including splash2 [98], parsec [99], NPB [100], and NU-MineBench [101]. The input set for the workloads is *simlarge*, unless state otherwise, shown in Table 7.3. The simulation collects the statistics only during the parallel parts of the workloads (the region of interest, ROI); the sequential sections at the beginning is used to warm up the cache and is excluded in the timing evaluation [102]. The reported the workloads execute more than 1 billion instructions in ROI and the workloads are sorted based on their memory intensity. The speedup of the workloads is normalized to the baseline system that uses MSC.

7.5 Results and Analysis

7.5.1 Overall Performance

This section analyzes the performance of systems for various designs, which is evaluated on a 16-core system running parallel benchmark suites. Figure 7.18 compares CANDY to memory-side cache (MSC), and also an impractical coherent DRAM cache that uses a

Table 7.3: Workloads for Multi-socket System Study: benchmark, suites, and input size

Name	Suites	Input
mg	NPB	Set C
ocean.cont	splash2	2050x2050 Grid
ocean.ncont	splash2	2050x2050 Grid
facesim	parsec	80,598 particles, 372,126 tetrahedra, 1 frame
water.nsq	splash2	20^3 Molecules
streamcluster	parsec	16,384 input points, 128 point dimensions
kmeans	NU-MineBench	10M elements, 9 dimension, 16 cluster
vips	parsec	2,662 x 5,500 pixels
radiosity	splash2	BF refinement =$1.5e^{-3}$
fluidanimate	parsec	300,000 particles, 5 frames
dedup	parsec	184 MB file size
fmm	splash2	256K Particles
barnes	splash2	256K Particles,Timestep= 0.25

64MB SRAM storage for the coherence directory and has zero L4 cache read latency for RFD operation (Impractical-CDC). Figure 7.18 also shows the performance for a system that has no DRAM cache (termed *L3-Only*). The baseline is memory-side cache, and the geometric mean is shown in the right most bar, labeled *AVG*. On average, L3-Only (i.e., no DRAM caches) degrades performance by 18%, with a maximum loss of 58% from *mg*. Embedded-CDir outperforms MSC by an average of 11%, but degrades performance for a couple of workloads (e.g., *streamcluster* and *radiosity*). CANDY not only mitigates such performance degradation but also improves average performance by 14% over Embedded-CDir. Overall, CANDY outperforms MSC by an average of 25% with a maximum improvement of 1.8X from *ocean.cont*. In addition, while Impractical-CDC has an average speedup of 30%, CANDY gets almost all the potential performance improvement, within 5% of the impractical case.

7.5.2 Sensitivity Studies: Scalability and Network Latency

One ket metric for a multi-socket system is its scalability when the number of socket increases. This section studies the sensitivity that varies the number of nodes by changing

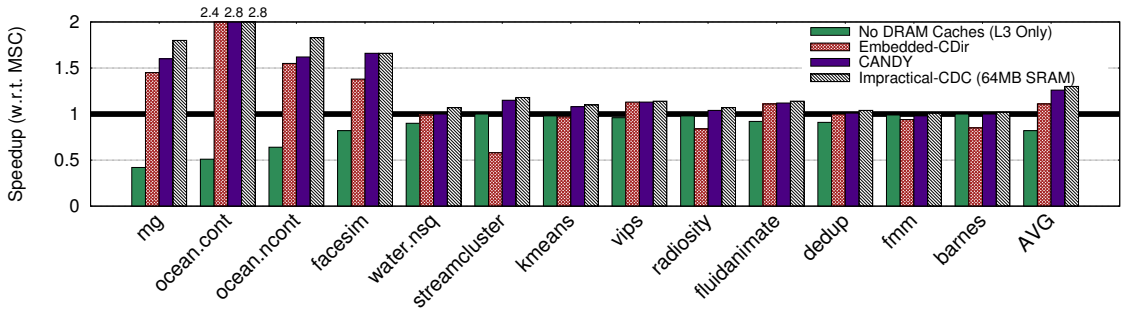

Figure 7.18: Performance of no-DRAM-cache design (*L3-Only*), Embedded-CDir, CANDY, and impractical CDC with 64MB SRAM overhead and zero L4 cache read latency for RFD operation (Impractical-CDC). Note that the performance is normalized to the baseline memory-side cache. The geometric mean, labeled as an *AVG*, is in the right most bar.

the total number of processors while keeping the number of processors per node constant: The number of nodes varies from 2 nodes to 8 nodes (8 cores to 32 cores), and the performance is shown for MSC and CANDY in Figure 7.19(a). Note that the speedup is with respect to each own baseline. CANDY outperforms MSC consistently across the spectrum and improves performance by 41%, 25%, 32% for 2-node, 4-node, and 8-node systems, respectively. Besides the scalability for the number of nodes, this section also studies the sensitivity to network latency. As CANDY saves the inter-node latency of cache misses to remote node, the inter-node latency is critical to the effectiveness of CANDY. Figure 7.19(b) shows a sensitivity study that varies the inter-node network latency from 0.5X (25ns) to 2X (100ns). CANDY consistently outperforms MSC by 25%, 25%, and 29% speedup for 0.5X, 1X, and 2X inter-node network latency, respectively. For long inter-node latency (100ns one-way), CANDY is more effective by providing 29% speedup.

7.5.3 Savings of Inter-Node Network Traffic

CDC enables the DRAM cache to keep the remote data; such capability not only avoids the inter-node network latency, but also reduces inter-node traffic and alleviates the bandwidth pressure on the inter-node network. Figure 7.20 shows the inter-node network traffic reduc-

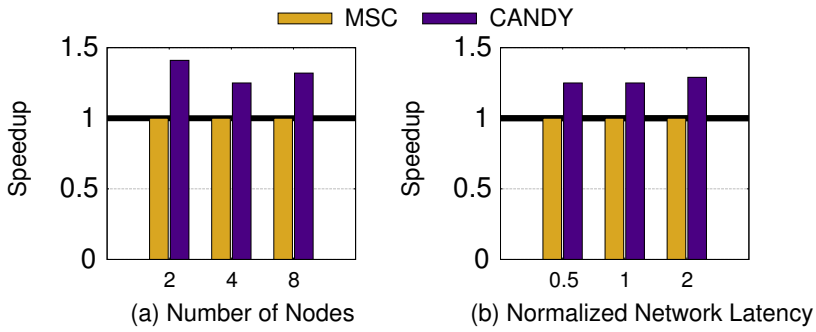

Figure 7.19: Sensitivity studies: (a) scalability with number of nodes and (b) normalized network latency (50ns as 1X).

tion by CANDY. (The figure is showing reduction, so the higher the better.) On average, CANDY reduces the traffic by 63%, meaning CANDY, compared to MSC, incurs only $\frac{1}{3}$ the traffic. Notice that workloads with significant traffic reduction tend to have significant performance improvement, as CANDY is able to cache the private data or read-only shared data of such workloads in the local DRAM caches.

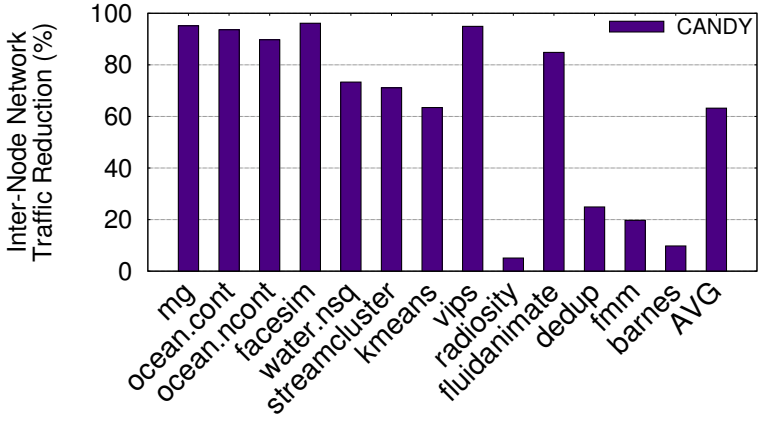

Figure 7.20: Inter-node network traffic reduction by CANDY (the higher the better).

7.5.4 Data Placement in Multi-Node Systems

To avoid inter-node network latency, prior work exploits thread-local data in non-uniform memory access (NUMA) systems [103]. Although the default memory mapping is interleaving among nodes, this section studies the implication of NUMA-aware systems [19, 18, 104]. In such systems, the operating system maps pages using NUMA-aware data place-

ment policy (e.g., *First-Touch*) [105, 106]. Figure 7.21 shows such systems for MSC and CANDY. Note that *CANDY (NUMA-Aware)* is normalized to MSC that also uses NUMA-aware policy. On average, CANDY (NUMA-Aware) outperforms MSC by 26%. Note that *streamcluster* prefers interleaved data placement, because its programming model appoints the master thread to initiate data structures, and centralizes all data in one node [107]. Figure 7.21 also shows a configuration where OS optimizes the page placement policy for individual workload based on the performance (termed *SW-Opt*). The OS chooses the best-performing page mapping policy for MSC, and uses the same policy for both MSC and CANDY. Even with such highly optimized data placement policy, CANDY (SW-Opt) still outperforms MSC by an average of 15%.

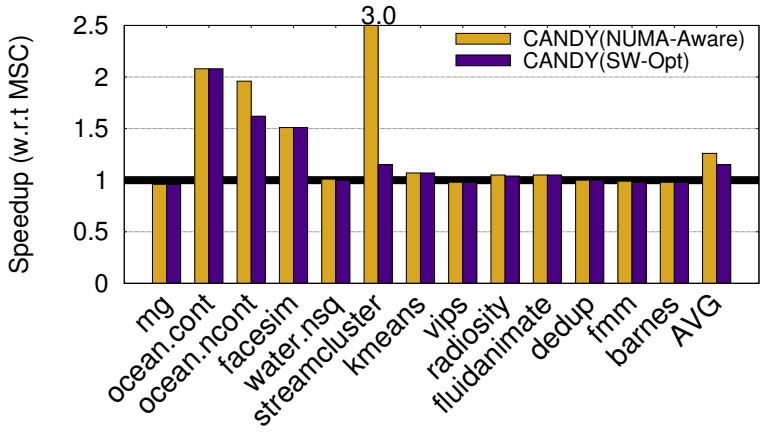

Figure 7.21: Performance of CANDY with NUMA-aware and software-optimized (SW-Opt) policy

7.6 Summary

This chapter studies DRAM caches for multi-node systems, in which each node has one DRAM cache. To architect giga-scale coherence DRAM caches, this chapter discovers two key challenges: (1) the coherence directory, whose size is as large as tens of MB, thus incurring prohibitive overheads of storage and latency and (2) the request-for-data operation, which is critical to access the most recent copy of data. This dissertation proposes CANDY, a scalable and low-cost solution to address both issues to enable high-performing CDC in

multi-node systems. First, to accommodate such a large structure, CANDY dedicates a portion of the 3D-DRAM capacity to avoid SRAM storage overhead. To mitigate the latency to access the coherence directory in 3D-DRAM, CANDY uses a DRAM-cache coherence buffer (DCB), which re-purposes the existing on-die coherence directory to cache recently accessed coherence directory entries. As the on-die coherence directory is already provisioned for L3 cache coherence, CANDY does not incur any SRAM storage overhead. Also, CANDY further exploits spatial locality to co-organize DCB and Embedded-CDir to improve the DCB hit rate.

Besides DCB, to mitigate the request-for-data latency for read-write shared data, the dissertation proposes sharing-aware bypass (SAB), which dynamically identifies read-write shared data, and enforces such data to bypass DRAM caches. The insight is that SAB mitigates the latency if read-write shared data is stored only in L3 caches. SAB is a simple mechanism that identifies the read-write shared data at run time and also enforce the bypassing decision for the system. SAB incurs negligible overheads of 8KB per node, but is effective to mitigate the request-for-data latency. The performance is evaluated by parallel workloads in a 4-node system. CANDY outperforms memory-side cache by 25%, with negligible overhead of 8KB per node; still, it provides within 5% of the potential performance improvement from an impractical coherent DRAM cache that incurs a storage overhead of 64MB SRAM with idealized request-for-data latency.

CHAPTER 8
CONCLUSION AND FUTURE WORK

8.1 Conclusion

Recent advancements in technology of die-stacking and packaging have enabled 3D-DRAM, which offers high memory bandwidth. Through integration of high-bandwidth 3D-DRAM and high-capacity memory technology such as commodity DRAM or non-volatile memory, heterogeneous memory systems can bridge the performance gap between processors and memory. However, conventional management techniques that are developed for on-chip SRAM caches are not suitable for heterogeneous memory systems; the techniques under-use the benefits delivered by 3D-DRAM and result in sub-optimal performance. To address the challenge posed by the heterogeneous memory system, this dissertation investigates the problems of hardware management for 3D-DRAM, resource utilization for a heterogeneous memory system, and scalability for systems with multiple 3D-DRAMs and proposes simple architectural solutions to improve the performance.

Chapter 3 analyzes the bandwidth bloat problem of DRAM caches, which results from the secondary operations that maintain the cache functionality. The dissertation proposes three orthogonal techniques, each of which reduces one component of the following operations: miss fill, writeback probe, and miss probe. The dissertation shows that the proposed technique reduces the bandwidth bloat by 32% and thus improves the DRAM cache access latency by 25%. Furthermore, the reduction of the bandwidth bloat and the improvement of the access latency result in 11% performance improvement.

Chapter 4 analyzes the trade-off for set-associative DRAM caches in non-volatile-memory-based heterogeneous memory systems. Set associativity in DRAM caches always pays the overhead of hit latency but not always delivers the benefit of hit-rate improve-

ment, which degrades performance for some workloads. The dissertation proposes an infrastructure that allows a low-cost transition between two degrees of set associativities and also a mechanism that dynamically chooses the best set associativities, which allow the DRAM cache to use set associativity only when useful and incur only negligible performance degradation.

Chapter 5 shows that the conventional wisdom, using only fast memory to service memory requests, is not suitable for heterogeneous memory systems with 3D-DRAM. This dissertation shows that the performance can be improved by a simple runtime mechanism that utilizes the aggregate system bandwidth. It demonstrates that the idea is applicable in two usage of heterogeneous memory systems and the effectiveness of the bandwidth utilization improves the performance by 11% in the cache mode and 10% in the flat mode.

Chapter 6 addresses the dilemma that two-level memory does not have software-transparent fine-grained data management and 3D-DRAM being a DRAM cache does not contribute to the memory capacity. This dissertation shows that a cache-like memory organization can have all benefits: the capacity of two-level memory and the fine-grained data management and software transparency of DRAM caches, thus improving the performance for a wide range of workloads. The dissertation proposes a line location table and a location predictor that achieves high accuracy with simple hardware.

Chapter 7 presents two key challenges that enables coherent DRAM caches for multi-socket systems: a large coherence directory and long-latency request-for-data operations. This dissertation shows that the performance can be improved by a novel architectural technique that reuses the on-die coherence directory and a sharing-aware bypass scheme that mitigates the latency overhead. The proposed techniques are robust to the data placement and various configurations in multi-socket systems.

8.2 Future Work

8.2.1 Improving Set-associative DRAM Caches

To deploy set associativity in DRAM caches, this dissertation takes the pay-as-you-go approach that incurs the overhead only when set associativity is useful. Another direction is to reduce the overhead such that set associativity becomes affordable. The overhead results from the need to locate the way of the requested cache line by either simultaneous reading all ways or sequentially reading one way after another. The way information can be speculated by a way predictor, which may mitigate the overhead and enable a low-cost set associative DRAM cache for PCM-based heterogeneous memory systems. However, under the current replacement policy (random), the challenge is to architect a simple and accurate way predictor. Also, another possible improvement of the set-associative DRAM cache is a low-overhead replacement policy. Tradition intelligent replacement policy developed for SRAM caches requires a status update on cache hits and misses, which incurs a huge bandwidth overhead for DRAM caches. Therefore, the challenge of update-free replacement policy for DRAM caches can be a huge step that further improves the performance of DRAM caches.

8.2.2 Coherence Directory of Coherent DRAM Caches

This dissertation uses sparse directory as the coherence protocol and proposes a two-level structure with the DRAM-cache coherence buffer to mitigate tens of megabyte storage for giga-scale DRAM caches. For this problem, one possible solution is to reduce the storage requirement such that the directory can fit the on-chip storage, eliminating the need of a two-level organization. One can exploit the locality in the content of directory and use compression techniques to reduce the storage; one can also exploit the characteristics of shared data and use a mix of directory-based and snoop-based protocols to lower the storage overhead of coherence directory.

8.2.3 Low-power Heterogeneous Memory Systems

The dissertation presents a set of bandwidth-efficient techniques that not only reduces the activities in the system but also improves the performance of the system, thus improves the energy consumption. However, the dissertation does not investigate techniques that enables low-power heterogeneous memory systems. Recent disclosed documents reveal that 3D-DRAM is more energy-efficient than commodity DRAM; therefore, the energy-proportionality can improve via data re-arrangement and power-down techniques. Also, 3D-DRAM being a cache provides the edge that part of cache can be disabled without the loss of data. The disabled part of the cache can be powered down to save energy. However, such studies need further examination of power-performance trade-offs

REFERENCES

[1] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *Sigarch comput. archit. news*, vol. 23, no. 1, pp. 20–24, Mar. 1995.

[2] D. Burger, J. R. Goodman, and A. Kägi, "Memory bandwidth limitations of future microprocessors," in *Proceedings of the 23rd annual international symposium on computer architecture*, ser. ISCA '96, Philadelphia, Pennsylvania, USA: ACM, 1996, pp. 78–89, ISBN: 0-89791-786-3.

[3] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin, "Scaling the bandwidth wall: Challenges in and avenues for cmp scaling," in *Proceedings of the 36th annual international symposium on computer architecture*, ser. ISCA '09, Austin, TX, USA: ACM, 2009, pp. 371–382, ISBN: 978-1-60558-526-0.

[4] Micron, *DDR3 spec*, 2010.

[5] JEDEC, *DDR4 spec (jesd79-4)*, 2013.

[6] HMCC, *HMC specification 1.0*, 2013.

[7] Micron, *HMC gen2*, Micron, 2014.

[8] J. Bolaria, "Micron reinvents DRAM memory," *Microprocessor report*, 2011.

[9] JEDEC, *High bandwidth memory (HBM) DRAM (jesd235)*, 2013, JEDEC.

[10] ——, *Wide I/O single data rate (WIDE I/O SDR)*, 2011.

[11] J. U. Knickerbocker, P. S. Andry, B. Dang, R. R. Horton, M. J. Interrante, C. S. Patel, R. J. Polastre, K. Sakuma, R. Sirdeshmukh, E. J. Sprogis, S. M. Sri-Jayantha, A. M. Stephens, A. W. Topol, C. K. Tsang, B. C. Webb, and S. L. Wright, "Three-dimensional silicon integration," *Ibm j. res. dev.*, vol. 52, no. 6, pp. 553–569, Nov. 2008.

[12] U. Kang, H.-J. Chung, S. Heo, S.-H. Ahn, H. Lee, S.-H. Cha, J. Ahn, D. Kwon, J. H. Kim, J.-W. Lee, H.-S. Joo, W.-S. Kim, H.-K. Kim, E.-M. Lee, S.-R. Kim, K.-H. Ma, D.-H. Jang, N.-S. Kim, M.-S. Choi, S.-J. Oh, J.-B. Lee, T.-K. Jung, J.-H. Yoo, and C. Kim, "8gb 3d ddr3 dram using through-silicon-via technology," 130–131,131a, 2009.

[13] AMD, *AMD radeon r9*, 2015.

[14] A. Sodani, "Knights landing (knl): 2nd generation intel xeon phi processor," in *2015 ieee hot chips 27 symposium (hcs)*, 2015, pp. 1–24.

[15] NVIDIA, *NVIDIA updates gpu roadmap; announces pascal*, 2015.

[16] J. Sim, A. R. Alameldeen, Z. Chishti, C. Wilkerson, and H. Kim, "Transparent hardware management of stacked dram as part of memory," in *Proceedings of the 47th annual ieee/acm international symposium on microarchitecture*, ser. MICRO-47, Cambridge, United Kingdom: IEEE Computer Society, 2014, pp. 13–24, ISBN: 978-1-4799-6998-2.

[17] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. H. Loh, "Heterogeneous memory architectures: A hw/sw approach for mixing die-stacked and off-package memories," in *2015 ieee 21st international symposium on high performance computer architecture (hpca)*, 2015, pp. 126–136.

[18] F. Dahlgren and J. Torrellas, "Cache-only memory architectures," *Computer*, vol. 32, no. 6, pp. 72–79, Jun. 1999.

[19] B. Falsafi and D. A. Wood, "Reactive numa: A design for unifying s-coma and cc-numa," in *Proceedings of the 24th annual international symposium on computer architecture*, ser. ISCA '97, Denver, Colorado, USA: ACM, 1997, pp. 229–240, ISBN: 0-89791-901-7.

[20] P. Machanick, P. Salverda, and L. Pompe, "Hardware-software trade-offs in a direct rambus implementation of the rampage memory hierarchy," 11, vol. 33, New York, NY, USA: ACM, Oct. 1998, pp. 105–114.

[21] G. H. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked dram caches," in *Proceedings of the 44th annual ieee/acm international symposium on microarchitecture*, ser. MICRO-44, Porto Alegre, Brazil: ACM, 2011, pp. 454–464, ISBN: 978-1-4503-1053-6.

[22] J. Sim, G. H. Loh, H. Kim, M. O'Connor, and M. Thottethodi, "A mostly-clean dram cache for effective hit speculation and self-balancing dispatch," in *Proceedings of the 2012 45th annual ieee/acm international symposium on microarchitecture*, ser. MICRO-45, Vancouver, B.C., CANADA: IEEE Computer Society, 2012, pp. 247–257, ISBN: 978-0-7695-4924-8.

[23] J. B. Rothman and A. J. Smith, "Sector cache design and performance," in *Proceedings 8th international symposium on modeling, analysis and simulation of computer and telecommunication systems (cat. no.pr00728)*, 2000, pp. 124–133.

[24] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked dram caches for servers: Hit ratio, latency, or bandwidth? have it all with footprint cache," in *Proceedings of the 40th*

annual international symposium on computer architecture, ser. ISCA '13, Tel-Aviv, Israel: ACM, 2013, pp. 404–415, ISBN: 978-1-4503-2079-5.

[25] S. Somogyi, T. F. Wenisch, A. Ailamaki, B. Falsafi, and A. Moshovos, "Spatial memory streaming," in *Proceedings of the 33rd annual international symposium on computer architecture*, ser. ISCA '06, Washington, DC, USA: IEEE Computer Society, 2006, pp. 252–263, ISBN: 0-7695-2608-X.

[26] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked dram cache," in *Proceedings of the 47th annual ieee/acm international symposium on microarchitecture*, ser. MICRO-47, Cambridge, United Kingdom: IEEE Computer Society, 2014, pp. 25–37, ISBN: 978-1-4799-6998-2.

[27] M. D. Powell, A. Agarwal, T. N. Vijaykumar, B. Falsafi, and K. Roy, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," in *Proceedings of the 34th annual acm/ieee international symposium on microarchitecture*, ser. MICRO 34, Austin, Texas: IEEE Computer Society, 2001, pp. 54–65, ISBN: 0-7695-1369-7.

[28] S. Franey and M. Lipasti, "Tag tables," in *2015 ieee 21st international symposium on high performance computer architecture (hpca)*, 2015, pp. 514–525.

[29] N. Gulur, M. Mehendale, R. Manikantan, and R. Govindarajan, "Bi-modal dram cache: A scalable and effective die-stacked dram cache," in *Proceedings of the 47th annual ieee/acm international symposium on microarchitecture*, ser. MICRO-47, Cambridge, United Kingdom: IEEE Computer Society, 2014, pp. 38–50, ISBN: 978-1-4799-6998-2.

[30] M. K. Qureshi and G. H. Loh, "Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design," in *Proceedings of the 2012 45th annual international symposium on microarchitecture*, 2012, ISBN: 978-0-7695-4924-8.

[31] A. Sodani, R. Gramunt, J. Corbal, H. S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. C. Liu, "Knights landing: Second-generation intel xeon phi product," *Ieee micro*, vol. 36, no. 2, pp. 34–46, 2016.

[32] M. M. K. Martin, M. D. Hill, and D. J. Sorin, "Why on-chip cache coherence is here to stay," *Commun. acm*, vol. 55, no. 7, pp. 78–89, Jul. 2012.

[33] A. Moshovos, "Regionscout: Exploiting coarse grain sharing in snoop-based coherence," in *Computer architecture, 2005. isca '05. proceedings. 32nd international symposium on*, 2005, pp. 234–245.

[34] J. F. Cantin, M. H. Lipasti, and J. E. Smith, "Improving multiprocessor performance with coarse-grain coherence tracking," in *Proceedings of the 32nd annual international symposium on computer architecture*, ser. ISCA '05, IEEE Computer Society, 2005, pp. 246–257, ISBN: 0-7695-2270-X.

[35] V. Salapura, M. Blumrich, and A. Gara, "Design and implementation of the blue gene/p snoop filter," in *High performance computer architecture, 2008. hpca 2008. ieee 14th international symposium on*, 2008, pp. 5–14.

[36] J. Zebchuk, V. Srinivasan, M. K. Qureshi, and A. Moshovos, "A tagless coherence directory," in *Proceedings of the 42nd annual ieee/acm international symposium on microarchitecture*, ser. MICRO 42, New York, New York: ACM, 2009, pp. 423–434, ISBN: 978-1-60558-798-1.

[37] A. Gupta, W. dietrich Weber, and T. Mowry, "Reducing memory and traffic requirements for scalable directory-based cache coherence schemes," in *In international conference on parallel processing*, 1990, pp. 312–321.

[38] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz, "An evaluation of directory schemes for cache coherence," in *Proceedings of the 15th annual international symposium on computer architecture*, ser. ISCA '88, Honolulu, Hawaii, USA: IEEE Computer Society Press, 1988, pp. 280–298, ISBN: 0-8186-0861-7.

[39] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy, "The directory-based cache coherence protocol for the dash multiprocessor," in *Proceedings of the 17th annual international symposium on computer architecture*, ser. ISCA '90, Seattle, Washington, USA: ACM, 1990, pp. 148–159, ISBN: 0-89791-366-3.

[40] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese, "Piranha: A scalable architecture based on single-chip multiprocessing," in *Proceedings of the 27th annual international symposium on computer architecture*, ser. ISCA '00, Vancouver, British Columbia, Canada: ACM, 2000, pp. 282–293, ISBN: 1-58113-232-8.

[41] Oracle, *OpenSPARC T2 overview*, 2013.

[42] M. Ferdman, P. Lotfi-Kamran, K. Balet, and B. Falsafi, "Cuckoo directory: A scalable directory for many-core systems," in *High performance computer architecture (hpca), 2011 ieee 17th international symposium on*, 2011, pp. 169–180.

[43] B. A. Cuesta, A. Ros, M. E. Gómez, A. Robles, and J. F. Duato, "Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks," in *Proceedings of the 38th annual international symposium on computer architecture*, ser. ISCA '11, San Jose, California, USA: ACM, 2011, pp. 93–104, ISBN: 978-1-4503-0472-6.

[44] D. Sanchez and C. Kozyrakis, "Scd: A scalable coherence directory with flexible sharer set encoding," in *Proceedings of the 2012 ieee 18th international symposium on high-performance computer architecture*, ser. HPCA '12, IEEE Computer Society, 2012, ISBN: 978-1-4673-0827-4.

[45] S. Demetriades and S. Cho, "Stash directory: A scalable directory for many-core coherence," in *High performance computer architecture (hpca), 2014 ieee 20th international symposium on*, 2014.

[46] M. E. Acacio, J. Gonzalez, J. M. Garcia, and J. Duato, "A two-level directory architecture for highly scalable cc-numa multiprocessors," *Ieee trans. parallel distrib. syst.*, vol. 16, no. 1, pp. 67–79, Jan. 2005.

[47] J. J. Valls, A. Ros, J. Sahuquillo, and M. E. Gómez, "Ps directory: A scalable multilevel directory cache for cmps," *J. supercomput.*, vol. 71, no. 8, pp. 2847–2876, Aug. 2015.

[48] L. Zhang, Z. Fang, and J. B. Carter, "Highly efficient synchronization based on active memory operations," in *Parallel and distributed processing symposium, 2004. proceedings. 18th international*, 2004, pp. 58–.

[49] J. H. Ahn, M. Erez, and W. J. Dally, "Scatter-add in data parallel architectures," in *High-performance computer architecture, 2005. hpca-11. 11th international symposium on*, 2005, pp. 132–142.

[50] H. Hoffmann, D. Wentzlaff, and A. Agarwal, "Remote store programming: A memory model for embedded multicore," in *Proceedings of the 5th international conference on high performance embedded architectures and compilers*, ser. HiPEAC'10, Pisa, Italy: Springer-Verlag, 2010, pp. 3–17, ISBN: 3-642-11514-4, 978-3-642-11514-1.

[51] G. Zhang, W. Horn, and D. Sanchez, "Exploiting commutativity to reduce the cost of updates to shared data in cache-coherent systems," in *Proceedings of the 48th international symposium on microarchitecture*, ser. MICRO-48, Waikiki, Hawaii: ACM, 2015, pp. 13–25, ISBN: 978-1-4503-4034-2.

[52] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," in *Proceedings of the 34th annual international symposium on computer architecture*, ser. ISCA '07, San Diego, California, USA: ACM, 2007, pp. 381–391, ISBN: 978-1-59593-706-3.

[53] D. A. Jiménez, "Insertion and promotion for tree-based pseudolru last-level caches," in *Micro-46*, 2013.

[54] A. Jaleel, K. B. Theobald, S. C. Steely Jr., and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," in *Proceedings of the 37th annual international symposium on computer architecture*, 2010, ISBN: 978-1-4503-0053-7.

[55] C.-J. Wu, A. Jaleel, W. Hasenplaugh, M. Martonosi, S. C. Steely Jr., and J. Emer, "Ship: Signature-based hit predictor for high performance caching," in *Proceedings of the 44th annual ieee/acm international symposium on microarchitecture*, ser. MICRO-44, Porto Alegre, Brazil: ACM, 2011, pp. 430–441, ISBN: 978-1-4503-1053-6.

[56] A.-C. Lai, C. Fide, and B. Falsafi, "Dead-block prediction & dead-block correlating prefetchers," in *Proceedings of the 28th annual international symposium on computer architecture*, ser. ISCA '01, Göteborg, Sweden: ACM, 2001, pp. 144–154, ISBN: 0-7695-1162-7.

[57] S. M. Khan, Y. Tian, and D. A. Jimenez, "Sampling dead block prediction for last-level caches," in *Proceedings of the 2010 43rd annual ieee/acm international symposium on microarchitecture*, ser. MICRO '43, Washington, DC, USA: IEEE Computer Society, 2010, pp. 175–186, ISBN: 978-0-7695-4299-7.

[58] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Proceedings of the 39th annual ieee/acm international symposium on microarchitecture*, ser. MICRO 39, Washington, DC, USA: IEEE Computer Society, 2006, pp. 423–432, ISBN: 0-7695-2732-9.

[59] S. M. Zahedi and B. C. Lee, "REF: Resource elasticity fairness with sharing incentives for multiprocessors," in *Proceedings of the 19th international conference on architectural support for programming languages and operating systems*, ser. ASPLOS '14, Salt Lake City, Utah, USA: ACM, 2014, pp. 145–160, ISBN: 978-1-4503-2305-5.

[60] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," in *Proceedings of the 27th annual international symposium on computer architecture*, ser. ISCA '00, Vancouver, British Columbia, Canada: ACM, 2000, pp. 128–138, ISBN: 1-58113-232-8.

[61] O. Mutlu and T. Moscibroda, "Stall-time fair memory access scheduling for chip multiprocessors," in *Proceedings of the 40th annual ieee/acm international symposium on microarchitecture*, ser. MICRO 40, Washington, DC, USA: IEEE Computer Society, 2007, pp. 146–160, ISBN: 0-7695-3047-8.

[62] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter, "Thread cluster memory scheduling: Exploiting differences in memory access behavior," in *Proceedings*

of the 2010 43rd annual ieee/acm international symposium on microarchitecture, ser. MICRO '43, Washington, DC, USA: IEEE Computer Society, 2010, pp. 65–76, ISBN: 978-0-7695-4299-7.

[63] A. Jaleel, E. Borch, M. Bhandaru, S. C. Steely Jr., and J. Emer, "Achieving non-inclusive cache performance with inclusive caches: Temporal locality aware (tla) cache management policies," in *Proceedings of the 2010 43rd annual international symposium on microarchitecture*, 2010, ISBN: 978-0-7695-4299-7.

[64] N. Chatterjee, R. Balasubramonian, M. Shevgoor, S. H. Pugsley, A. N. Udipi, A. Shafiee, K. Sudan, and M. Awasthi, *USIMM*, University of Utah, 2012.

[65] J. L. Henning, "Spec cpu2006 benchmark descriptions," *Sigarch comput. archit. news*, vol. 34, no. 4, pp. 1–17, Sep. 2006.

[66] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder, "Using simpoint for accurate and efficient simulation," in *Proceedings of the 2003 acm sigmetrics international conference on measurement and modeling of computer systems*, 2003, ISBN: 1-58113-664-1.

[67] J. Sim, G. H. Loh, H. Kim, M. O'Connor, and M. Thottethodi, "A mostly-clean dram cache for effective hit speculation and self-balancing dispatch," in *Proceedings of the 2012 45th annual international symposium on microarchitecture*, 2012, ISBN: 978-0-7695-4924-8.

[68] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y. c. Chen, R. M. Shelby, M. Salinga, D. Krebs, S. h. Chen, H. l. Lung, and C. H. Lam, "Phase-change random access memory: A scalable technology," *Ibm j. res. dev.*, vol. 52, no. 4, Jul. 2008.

[69] C. Chou, A. Jaleel, and M. K. Qureshi, "Bear: Techniques for mitigating bandwidth bloat in gigascale dram caches," in *Proceedings of the 42nd annual international symposium on computer architecture*, ser. ISCA '15, Portland, Oregon: ACM, 2015, pp. 198–210, ISBN: 978-1-4503-3402-0.

[70] J. D. McCalpin, *Stream: Sustainable memory bandwidth in high performance computer*, 1991.

[71] N. Agarwal, D. Nellans, M. Stephenson, M. O'Connor, and S. W. Keckler, "Page placement strategies for gpus within heterogeneous memory systems," 1, vol. 43, New York, NY, USA: ACM, Mar. 2015, pp. 607–618.

[72] S. De, R. Stewart, G. Cascaval, and D. Chun, *System and method for allocating memory to dissimilar memory devices using quality of service*, US Patent 9,092,327, 2015.

[73] M. A. Holliday, "Reference history, page size, and migration daemons in local/remote architectures," in *Proceedings of the third international conference on architectural support for programming languages and operating systems*, ser. ASPLOS III, Boston, Massachusetts, USA: ACM, 1989, pp. 104–112, ISBN: 0-89791-300-0.

[74] Intel, *Intel core i7 processor*, 2013.

[75] D. Kaseridis, J. Stuecheli, and L. K. John, "Minimalist open-page: A dram page-mode scheduling policy for the many-core era," in *Proceedings of the 44th annual ieee/acm international symposium on microarchitecture*, ser. MICRO-44, Porto Alegre, Brazil: ACM, 2011, pp. 24–35, ISBN: 978-1-4503-1053-6.

[76] G. H. Loh, N. Jayasena, J. Chung, S. K. Reinhardt, M. O'Connor, and K. McGrath, "Challenges in heterogeneous die-stacked and off-chip memory systems," in *3rd workshop on socs, heterogeneous architectures and workloads*, 2012.

[77] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 acm sigplan conference on programming language design and implementation*, 2005, ISBN: 1-59593-056-6.

[78] J. L. Henning, "Spec cpu2006 memory footprint," *Sigarch comput. archit. news*, vol. 35, no. 1, pp. 84–89, Mar. 2007.

[79] D. Gove, "Cpu2006 working set size," *Sigarch comput. archit. news*, vol. 35, no. 1, pp. 90–96, Mar. 2007.

[80] K. Tran, "The era of high bandwidth memory," in *2016 ieee hot chips 28 symposium (hcs)*, 2016.

[81] Micron, *Tn-46-03 calculating DDR memory system power*.

[82] T. Vogelsang, "Understanding the energy consumption of dynamic random access memories," in *Proceedings of the 2010 43rd annual ieee/acm international symposium on microarchitecture*, ser. MICRO '43, Washington, DC, USA: IEEE Computer Society, 2010, pp. 363–374, ISBN: 978-0-7695-4299-7.

[83] A. Hartstein, V. Srinivasan, T. R. Puzak, and P. G. Emma, "Cache miss behavior: Is it sqrt(2)?" In *Proceedings of the 3rd conference on computing frontiers*, 2006.

[84] Intel, *Intel ssd*.

[85] E. Seo, S. Y. Park, and B. Urgaonkar, "Empirical analysis on energy efficiency of flash-based ssds," in *Proceedings of the 2008 conference on power aware comput-*

ing and systems, ser. HotPower'08, San Diego, California: USENIX Association, 2008, pp. 17–17.

[86] SanDisk, *SanDisk solid state drive*.

[87] AMD, *Introduction to magny-cours*, 2010.

[88] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, and B. Hughes, "Cache hierarchy and memory subsystem of the amd opteron processor," *Micro, ieee*, vol. 30, no. 2, pp. 16–29, 2010.

[89] D. J. Sorin, M. D. Hill, and D. A. Wood, *A primer on memory consistency and cache coherence*, 1st. Morgan & Claypool Publishers, 2011, ISBN: 1608455645, 9781608455645.

[90] Intel, *Intel quickpath interconnect*, 2009.

[91] AMD, *AMD hypertransport*, 2001.

[92] P. J. Denning and S. C. Schwartz, "Properties of the working-set model," *Commun. acm*, vol. 15, no. 3, pp. 191–198, Mar. 1972.

[93] T. L. Johnson, M. C. Merten, and W.-M. W. Hwu, "Run-time spatial locality detection and optimization," in *Proceedings of the 30th annual acm/ieee international symposium on microarchitecture*, ser. MICRO 30, Research Triangle Park, North Carolina, USA: IEEE Computer Society, 1997, pp. 57–64, ISBN: 0-8186-7977-8.

[94] J. Weinberg, M. O. McCracken, E. Strohmaier, and A. Snavely, "Quantifying locality in the memory access patterns of hpc applications," in *Proceedings of the 2005 acm/ieee conference on supercomputing*, ser. SC '05, IEEE Computer Society, 2005, pp. 50–, ISBN: 1-59593-061-2.

[95] C. H. Chi and H. Dietz, "Improving cache performance by selective cache bypass," in *System sciences, 1989. vol.i: Architecture track, proceedings of the twenty-second annual hawaii international conference on*, vol. 1, 1989, 277–285 vol.1.

[96] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An evaluation of high-level mechanistic core models," *Acm trans. archit. code optim.*, vol. 11, no. 3, 28:1–28:25, Aug. 2014.

[97] M. S. Papamarcos and J. H. Patel, "A low-overhead coherence solution for multiprocessors with private cache memories," in *Proceedings of the 11th annual international symposium on computer architecture*, ser. ISCA '84, ACM, 1984, pp. 348–354, ISBN: 0-8186-0538-3.

[98] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," in *Proceedings of the 22nd annual international symposium on computer architecture*, ser. ISCA '95, S. Margherita Ligure, Italy: ACM, 1995, pp. 24–36, ISBN: 0-89791-698-0.

[99] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on parallel architectures and compilation techniques*, ser. PACT '08, Toronto, Ontario, Canada: ACM, 2008, pp. 72–81, ISBN: 978-1-60558-282-5.

[100] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The nas parallel benchmarks; summary and preliminary results," in *Proceedings of the 1991 acm/ieee conference on supercomputing*, ser. Supercomputing '91, Albuquerque, New Mexico, USA: ACM, 1991, pp. 158–165, ISBN: 0-89791-459-7.

[101] J. Pisharath, Y. Liu, W. keng Liao, A. Choudhary, G. Memik, and J. Parhi, "Numinebench 2.0," Center for Ultra-Scale Computing and Information Security, Northwestern University, Tech. Rep. CUCIS-2005-08-01, 2005.

[102] W. Heirman, S. Sarkar, T. E. Carlson, I. Hur, and L. Eeckhout, "Power-aware multicore simulation for early design stage hardware/software co-optimization," in *Proceedings of the 21st international conference on parallel architectures and compilation techniques*, ser. PACT '12, Minneapolis, Minnesota, USA: ACM, 2012, pp. 3–12, ISBN: 978-1-4503-1182-3.

[103] A. Roy and T. M. Jones, "Allarm: Optimizing sparse directories for thread-local data," in *Proceedings of the conference on design, automation & test in europe*, ser. DATE '14, Dresden, Germany: European Design and Automation Association, 2014, 78:1–78:6, ISBN: 978-3-9815370-2-4.

[104] P. Stenström, T. Joe, and A. Gupta, "Comparative performance evaluation of cache-coherent numa and coma architectures," in *Proceedings of the 19th annual international symposium on computer architecture*, ser. ISCA '92, Queensland, Australia: ACM, 1992, pp. 80–91, ISBN: 0-89791-509-7.

[105] Linux, *Linux 3.8 automatic numa balancing*, 2013.

[106] Microsoft, *Microsoft windows numa support*.

[107] F. Gaud, B. Lepers, J. Funston, M. Dashti, A. Fedorova, V. Quéma, R. Lachaize, and M. Roth, "Challenges of memory management on modern numa systems," *Commun. acm*, vol. 58, no. 12, pp. 59–66, Nov. 2015.